ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Дослідження шляхів та розробка рекомендацій щодо пошуку вразливостей вебзастосунків на базі методу SAST»

зі спеціальності *125 Кібербезпека*
(код, найменування спеціальності)

освітньо-професійної програми *Інформаційна та кібернетична безпека*
(назва програми)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Анастасія БРИГИНЕЦЬ
(підпис)

Виконала: здобувачка вищої освіти групи БСД-41
БРИГИНЕЦЬ Анастасія
(прізвище, ім'я)

Керівник д.техн.н., професор ГАЙДУР Галина
(науковий ступінь, вчене звання, прізвище, ім'я)

Рецензент _____
(науковий ступінь, вчене звання, прізвище, ім'я)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

Кафедра Інформаційної та кібернетичної безпеки
Ступінь вищої освіти Бакалавр
Спеціальність 125 Кібербезпека
Освітньо-професійна програма Інформаційна та кібернетична безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Галина ГАЙДУР
“___” _____ 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бригинець Анастасії Андріївні
(прізвище, ім’я)

1. Тема кваліфікаційної роботи: «Дослідження шляхів та розробка рекомендацій щодо пошуку вразливостей вебзастосунків на базі методу SAST»

керівник кваліфікаційної роботи Гайдур Галина, д.техн.н., професор
(прізвище, ім’я, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «_27_» _лютого_ 2024 року № __36__.

2. Строк подання здобувачкою вищої освіти кваліфікаційної роботи 31.05.2024 р.

3. Вихідні дані до кваліфікаційної роботи
програмний комплекс для віртуалізації;
рішення Snyk Code;
наукова та технічна література, експлуатаційна документація, нормативні документи, міжнародні стандарти.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз проблематики виявлення загроз у вебзастосунках.
2. Аналіз наявних рішень напрямку SAST, детальний огляд рішення Snyk Code
3. Розробка рекомендацій щодо розгортання SAST рішень для організації

5. Перелік графічного матеріалу
Презентація PowerPoint.

6. Дата видачі завдання 15.04.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ зп	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Визначення актуальності проблеми виявлення вразливостей у вебзастосунках	27.02.2024 р.	
2.	Аналіз наукової та технічної літератури з питань теми кваліфікаційної роботи.		
3.	Аналіз проблематики виявлення загроз у вебзастосунках		
4.	Аналіз наявних рішень напрямку SAST, детальний огляд рішення Snyk Code		
5.	Розробка рекомендацій щодо розгортання SAST рішень для організації.		
6.	Оформлення результатів дослідження.		
7.	Підготовка доповіді до захисту.	31.05.2024 р.	

Здобувачка вищої освіти _____ Анастасія БРИГИНЕЦЬ
(підпис) *(ім'я, прізвище)*

Керівник кваліфікаційної роботи _____ Галина ГАЙДУР
(підпис) *(ім'я, прізвище)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу

здобувачки БРИГИНЕЦЬ Анастасії

на тему: «Дослідження шляхів та розроблення рекомендацій щодо пошуку вразливостей вебзастосунків на базі методу SAST»

Актуальність: Забезпечення безпеки коду вебзастосунків є критично важливим у сучасних умовах зростання кіберзагроз. Статичний аналіз безпеки застосунків (SAST) дозволяє своєчасно виявляти вразливості до того, як зловмисники зможуть їх використати. Це знижує ризики несанкціонованого доступу та витоку інформації, забезпечуючи безперервність роботи та стійкість бізнесу до кібератак.

Актуальність теми дипломної роботи визначається постійною потребою у застосуванні найновіших технологій для зміцнення кібербезпеки.

Позитивні сторони:

1. Робота забезпечує детальне дослідження проблеми пошуку вразливостей у вебзастосунках, чітко формулюючи цілі та задачі застосування статичного аналізу безпеки застосунків (SAST).
2. Методи і засоби статичного аналізу безпеки застосунків ретельно проаналізовані з акцентом на використанні інструменту Snyk.
3. Представлено розроблені рекомендації щодо інтеграції SAST у процеси розробки для підвищення безпеки програмного коду вебзастосунків.
4. Текст роботи чітко структурований, містить послідовне і грамотне викладення матеріалу. Висновки сформульовані змістовно і чітко. Графічний матеріал та список використаних джерел свідчать про глибоке розуміння теми та вміння працювати з науково-технічною літературою.
5. Ефективність розроблених рекомендацій підтверджена через експериментальний аналіз за допомогою Snyk, демонструючи практичну значущість дослідження у вдосконаленні процесів безпеки програмного коду для вебзастосунків.

Недоліки:

1. Аналіз реального застосування SAST на прикладі конкретної організації міг би надати більш практичну перспективу та допомогти в глибшому розумінні ефективності запропонованих методів.
2. У роботі могла б бути більш детально розроблена методика інтеграції SAST з іншими інструментами безпеки в контексті SDLC, що дозволило б виявити потенційні перешкоди та визначити оптимальні підходи для різних типів проектів.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи.

Висновок: Враховуючи недоліки, кваліфікаційна робота заслуговує оцінку **“відмінно”**, а здобувачка **БРИГИНЕЦЬ Анастасія** – присвоєння кваліфікації бакалавра з кібербезпеки за освітньою програмою Інформаційна та кібернетична безпека

Рецензент:

_____	_____	_____
(науковий ступінь, вчене звання)	*(підпис)*	*(ім’я, прізвище)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на здобуття освітнього ступеня бакалавра

Направляється здобувачка БРИГИНЕЦЬ Анастасія до захисту кваліфікаційної роботи
(прізвище, ім'я)

спеціальності 125 Кібербезпека
освітньо-професійної програми Інформаційна та кібернетична безпека
(шифр і назва спеціальності)

на тему: «Дослідження шляхів та розроблення рекомендацій щодо пошуку вразливостей вебзастосунків на базі методу SAST».

Кваліфікаційна робота і рецензія додаються.

Директор інституту _____ Віталій САВЧЕНКО
(підпис) *(ім'я, прізвище)*

Висновок керівника кваліфікаційної роботи

Здобувачка ***БРИГИНЕЦЬ Анастасія*** вибрала для своєї роботи тему, спрямовану на дослідження методів статичного аналізу безпеки застосунків (SAST) та розроблення рекомендацій для виявлення вразливостей у вебзастосунках. Вибір і опрацювання наукових джерел свідчать про її глибоке розуміння теми та здатність застосовувати теоретичні знання на практиці. Протягом всього процесу виконання дипломної роботи ***БРИГИНЕЦЬ Анастасія*** проявила серйозний підхід, самостійність у розв'язанні складних питань і вчасно виконала всі заплановані завдання.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувачки ***БРИГИНЕЦЬ Анастасії*** на оцінку **"відмінно"** та присвоїти їй кваліфікацію бакалавр з кібербезпеки за освітньою програмою Інформаційна та кібернетична безпека.

Керівник кваліфікаційної роботи _____ Галина ГАЙДУР
(підпис) *(ім'я, прізвище)*
"____" _____ 2024 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувачка **БРИГИНЕЦЬ Анастасія** допускається до захисту даної кваліфікаційної роботи в Екзаменаційній комісії.

Завідувач кафедри Інформаційної та кібернетичної безпеки
(назва)

_____ Галина ГАЙДУР
(підпис) *(ім'я, прізвище)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи: 72 сторінки, 52 рисунки, 37 джерел.

Об'єкт дослідження – процеси пошуку вразливостей вебзастосунків.

Предмет дослідження – методи та засоби виявлення вразливостей в програмному коді вебзастосунків на базі статичного аналізу безпеки програмного коду.

Методи дослідження – опрацювання технічної літератури та наукових джерел для з'ясування основних принципів та переваг використання SAST та, зокрема, рішення Snyk; аналіз документації та випадків використання Snyk для забезпечення безпеки застосунків; проведення експериментального аналізу з використанням Snyk на тестових проєктах для оцінки ефективності виявлення вразливостей; порівняльний аналіз рішень для визначення унікальних переваг Snyk.

Статичний аналіз безпеки застосунку (SAST) є важливою компонентою в системі захисту вебзастосунків, спрямованою на виявлення вразливостей ще на ранніх етапах розробки. Використовуючи різні методи аналізу програмного коду без його виконання, SAST дозволяє ідентифікувати потенційні зломи та витоки даних. Особлива увага у дослідженні приділена застосуванню інструментів SAST на прикладі рішення Snyk, яке інтегрується безпосередньо в процеси розробки та надає детальний аналіз з мінімальними затримками в реальному часі, що сприяє підвищенню загального рівня безпеки продуктів.

Дослідження охоплює методи і засоби застосування SAST, з особливим акцентом на рішенні Snyk, для підвищення безпеки програмного коду. Аналізовано технічну літературу та наукові джерела, проведено експериментальний аналіз за допомогою Snyk, і виявлено переваги застосування цього рішення порівняно з альтернативними методами.

В результаті дослідження розроблено рекомендації щодо інтеграції SAST у процеси розробки, спрямовані на забезпечення вищого рівня безпеки

вебзастсоунків. Практичне застосування результатів роботи може знайти використання у сферах, де критично важливим є забезпечення надійності і безпе програмного забезпечення.

Галузь використання – кібербезпека інформаційних ресурсів організації.

КІБЕРБЕЗПЕКА, SAST, БЕЗПЕКА ЗАСТОСУНКІВ, SDLC, АНАЛІЗ КОДУ, СКАНЕР ВРАЗЛИВОСТЕЙ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ БЕЗПЕКИ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CI/CD – Continuous Integration/Continuous Development

CLI – Command Line Interface

DAST – Dynamic Application Security Testing

IDE – Integrated Development Environment

RCE – Remote Code Execution

SAST – Static Application Security Testing

SCA – Software Composition Analysis

SDLC – Software Development Lifecycle

XSS – Cross-Site Scripting

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ВСТУП

Актуальність дослідження. У сучасному цифровому світі, де більшість процесів у бізнесі, науці, освіті, та повсякденному житті залежать від використання вебзастосунків, питання організації їх безпеки набуває особливої актуальності. Вразливості в програмному коді можуть призвести не лише до втрати критично важливих даних, фінансових та репутаційних збитків, але й до серйозних загроз національній безпеці. У національному контексті, Україна – держава, яка щоденно стикається з численними атаками, тож питання кібербезпеки стає ще більш актуальним, вимагаючи розробки та впровадження передових методів захисту вебзастосунків.

Технологія статичного тестування безпеки застосунків (SAST) виступає одним з ключових превентивних інструментів від реалізації атак на вебзастосунки, дозволяючи ідентифікувати потенційні вразливості на ранніх етапах розробки. Програмне рішення Snyk, як один із провідних інструментів у цій області, надає можливості не тільки для виявлення вразливостей, але й для їх автоматизованого усунення, інтегруючись з сучасними підходами до розробки програмного забезпечення.

Проте, попри наявність ефективних інструментів та методик, питання інтеграції SAST у процеси розробки та забезпечення безперервної безпеки вебзастосунків залишається недостатньо дослідженим. Це створює прогалини у алгоритмах захисту, особливо в умовах динамічного розвитку програмного забезпечення та зростання кількості кіберзагроз. В цьому контексті, дослідження, спрямоване на аналіз та розробку рекомендацій щодо ефективного використання Snyk, як SAST рішення стає актуальним.

Робота над цією темою дозволить не тільки систематизувати наявні знання та найкращі практики використання Snyk, як SAST рішення в організації, але й виявити прогалини у цих процесах, запропонувавши шляхи їх оптимізації та інтеграції у різноманітні етапи розробки вебзастосунків. Таким чином, це дослідження не тільки корелюється із загальною проблематикою реалізації заходів

кібербезпеки, але й спрямоване на вирішення конкретних завдань, що стоять перед розробниками та фахівцями з кібербезпеки уже сьогодні.

Об'єкт дослідження – процеси пошуку вразливостей вебзастосунків.

Предмет дослідження – методи та засоби виявлення вразливостей в програмному коді вебзастосунків на базі статичного аналізу безпеки застосунків

Мета роботи – розробити рекомендації щодо застосування методів та засобів пошуку вразливостей у вебзастосунках на базі метода SAST та рекомендації щодо їх реалізації

Наукові завдання:

дослідити сутність проблеми пошуку вразливостей у вебзастосунках;

проаналізувати основні методики виявлення вразливостей у вебзастосунках;

проаналізувати існуючі рішення для виявлення вразливостей у вебзастосунках;

проаналізувати методи та засоби пошуку вразливостей у вебзастосунках на базі рішення Snyk;

запропонувати варіант алгоритму пошуку вразливостей на базі Snyk;

розробити рекомендації щодо застосування методів та засобів пошуку вразливостей у вебзастосунках.

Методи дослідження – опрацювання тематичної літератури, інтернет-джерел, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння.

Практичне значення одержаних результатів: розроблено рекомендації щодо застосування методів та засобів пошуку вразливостей у вебзастосунках, а також розроблено рекомендації фахівцям з кібербезпеки щодо їх реалізації.

Результати кваліфікаційної роботи апробовані на Всеукраїнській науковій конференції «Цифрова трансформація кібербезпеки», яка відбулася 26 квітня 2024 року в Державному університеті інформаційно-комунікаційних технологій, м. Київ.

1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ПОШУКУ ВРАЗЛИВОСТЕЙ У ВЕБЗАСТОСУНКАХ

1.1. Роль і місце вебзастосунків у організації

У стрімку цифрову епоху бізнесу необхідно йти в ногу з останніми технологічними досягненнями, щоб залишатися конкурентоспроможним на ринку. Одним з таких досягнень є розробка вебзастосунків. Вона може допомогти організаціям поширити свою присутність в Інтернеті, оптимізувати процеси, краще взаємодіяти зі своїми клієнтами, ефективніше управляти даними та заощаджувати кошти [1].

Згідно звіту Facts and Factors [2], вебзастосунки, будучи легкими у розгортанні та широко використовуваними серед користувачів по всьому світу, очікують середньорічнього зростання ринку близько 34% у період з 2020 по 2026 рік (рисунок 1.1).

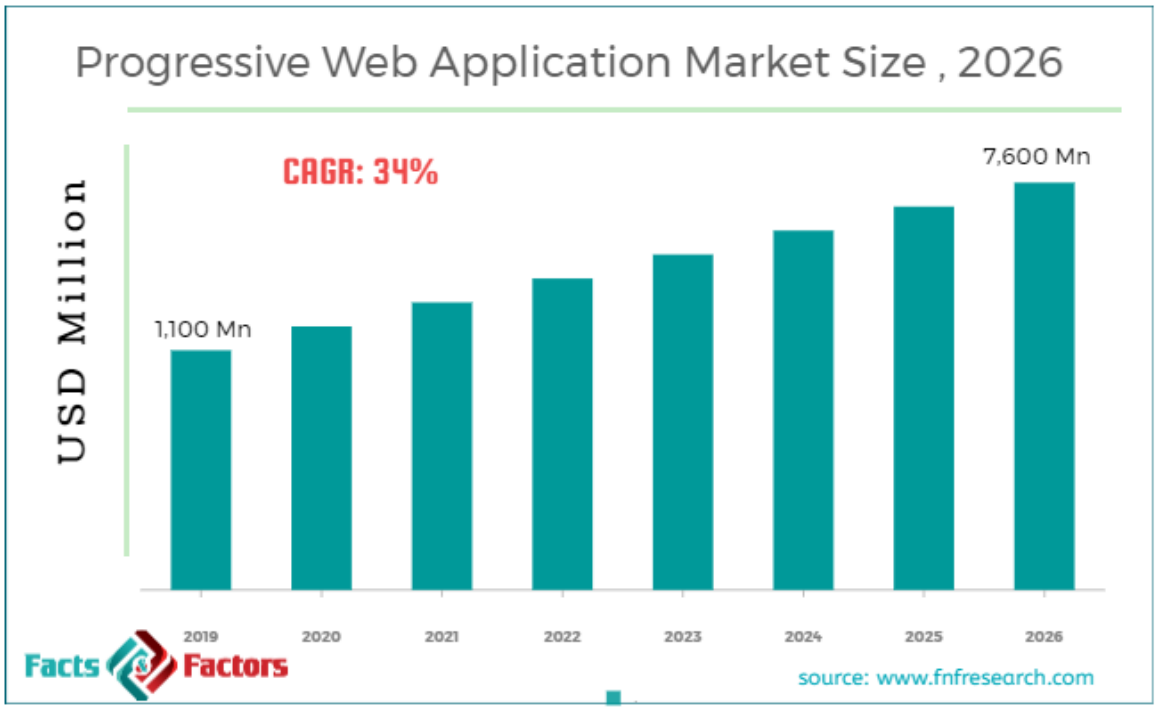


Рис. 1.1. Прогнозоване зростання ринку вебзастосунків [1]

Використання вебзастосунків у рамках робочих процесів організацій має ряд суттєвих переваг [3]:

1. Цілодобова доступність. Щоб взаємодіяти з вебзастосунком, користувачам потрібен лише ПК, ноутбук або смартфон з доступом до інтернету. Наприклад, відвідувачі інтернет-магазину можуть робити замовлення на сайті в будь-який час, не залежачи від графіку роботи продавців-консультантів, як це зазвичай буває в офлайн-торгівлі. Використовуючи такий інструмент, як вебзастосунок, можна забути про географічні обмеження і зробити бізнес по-справжньому глобальним. Крім того, такий підхід відкриває доступ до потенціалу SEO-оптимізації та багаторазових можливостей цифрового маркетингу.

2. Економічно ефективна розробка. На відміну від мобільної розробки, при розробці вебзастосунків не потрібно замовляти кілька версій для різних операційних систем. Користувачі взаємодіють з продуктом через єдиний інтерфейс для всіх пристроїв, наприклад, через бравзер. Ця особливість значно прискорює розробку і знижує її вартість.

3. Легкість доступу до бізнес-даних. Дані, якими користувачі обмінюються з вебзастосунком, зберігаються на віддаленому сервері (хмарному сховищі), до якого певні особи можуть отримати доступ у будь-який час і з будь-якого пристрою з доступом до інтернету. Дані доступні в режимі реального часу, і кілька користувачів можуть запитувати їх одночасно. Крім того, знижується ризик втрати даних через збої в роботі обладнання, оскільки дані надійно зберігаються в хмарному сховищі та створюються резервні копії.

4. Оптимізація процесів. Програмні системи, організовані у вигляді вебзастосунків, дозволяють автоматизувати численні ручні процеси, підвищити продуктивність працівників і скоротити витрати. Користувачі можуть отримати доступ до даних у режимі реального часу або у вигляді інформаційних панелей, що прискорює прийняття рішень і підвищує їхню якість (рисунок 1.2). Такий підхід може забезпечити значну конкурентну перевагу для будь-якого бізнесу.

Рис. 1.2. Використання вебзастосунку для управління кадрами [4]

5. Вебзастосунок легко масштабується. Застосунки у корпоративній сфері мають ще одну важливу перевагу - вони дуже гнучкі з точки зору масштабування. Якщо потрібно підключити більше співробітників, все, що потрібно - це лише трохи більше серверних ресурсів. Оскільки бізнес зростає і змінюється, програмне забезпечення, яке використовується, повинно йти в ногу з часом.

До того ж, веб-застосунки не потрібно завантажувати, оскільки доступ до них здійснюється через мережу. Користувачі можуть отримати доступ до вебзастосунків через бравзер, наприклад, Google Chrome, Mozilla Firefox або Safari [5].

Вебзастосунок поєднує в собі різні технології та мови програмування, щоб створити єдиний досвід для користувачів, які отримують доступ до нього через бравзер. Застосунок запускається на віддаленому сервері, часто відомому як вебсервер, і взаємодіє з користувачем через інтерфейс [6].

Вебзастосунки зазвичай мають короткі цикли розробки і невеликі команди розробників. Більшість вебзастосунків пишуться на JavaScript, HTML5 або CSS. Програмування на стороні клієнта зазвичай використовує мови, які допомагають

створити інтерфейс програми. Програмування на стороні сервера створює скрипти, які використовуватиме вебзастосунок. Такі мови, як Python, Java і Ruby, зазвичай використовуються для програмування на стороні сервера [5].

Коли користувач запитує доступ до вебзастосунку, запит перенаправляється на вебсервер, який отримує необхідні дані і повертає їх у браузер користувача. Згодом дані рендерингуються і відображаються користувачеві в бравзері.

Інтерфейс вебзастосунку зазвичай складається з HTML, CSS і JavaScript. HTML забезпечує каркас веб-сторінки, CSS її стилізує, а JavaScript додає взаємодії з веб-сайтом, роблячи його більш динамічним і зручним для користувача.

Внутрішня частина вебзастосунку відповідає за логіку на стороні сервера, управління базами даних і налаштування сервера. Для прискорення розробки використовуються такі фреймворки, як Laravel, Django, Ruby on Rails і Spring, а також різноманітні мови програмування, такі як PHP, Python, Ruby і Java.

Вебзастосунок складається з фронтенду та бекенду, де фронтенд керує взаємодією з користувачем, а бекенд відповідає за логіку на стороні сервера та адміністрування даних. Застосунок працює на віддаленому сервері і взаємодіє з користувачем через бравзер, що забезпечує безперебійну і швидку роботу користувача. [6]

Бекенд вебзастосунків взаємодіє з фронтендом за допомогою інтерфейсів прикладного програмування (API) та вебсервісів. Ці інтерфейси дозволяють передавати дані між інтерфейсом і бекендом, що робить вебзастосунки динамічними та адаптивними.

Важливо зазначити, що розробка вебзастосунків має свій певний життєвий цикл – SDLC.

SDLC (Software development lifecycle) – це процес, який використовується індустрією програмного забезпечення для проєктування, розробки і тестування високоякісного програмного забезпечення (рис. 1.3).

SDLC націлений на виробництво ПЗ, яке відповідає очікуванням клієнтів або перевершує їх, в найкоротші терміни завершує роботу і оцінює витрати. Цей процес складається з шести основних фаз [7].

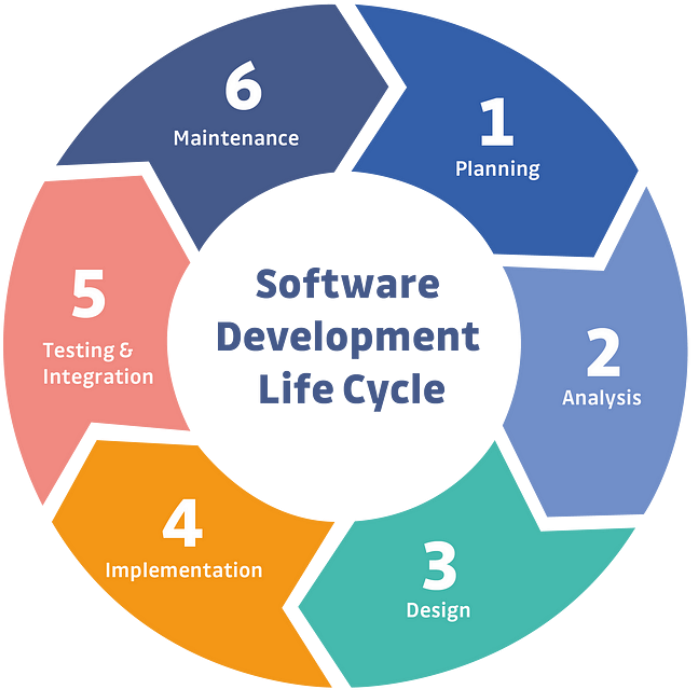


Рис. 1.3. Життєвий цикл розробки програмного забезпечення [8]

1. Планування системи. Фаза планування – найбільш важливий і критичний крок у створенні успішної системи. На цьому етапі:

• точно вирішується що саме потрібно зробити, розробити, які проблеми вирішити, які потреби закрити;

• визначаються проблеми, цілі і ресурси (таких, як персонал і витрати);

• вивчаються можливості альтернативних рішень шляхом зустрічей з клієнтами, постачальниками, консультантами та співробітниками;

• вивчається, як зробити продукт краще, ніж у конкурентів.

Після аналізу цих даних є три варіанти: розробити нову систему, покращити існуючу або залишити систему як є. На цьому етапі надзвичайно важлива комунікація з замовником.

2. Аналіз системи. Необхідно визначити і задокументувати вимоги кінцевого користувача системи – в чому його очікування і як їх здійснити.

Крім того, для проєкту робиться техніко-економічне обґрунтування, яке з'ясовує, чи є проект організаційно, економічно, соціально, технологічно здійсненним.

Дуже важливо підтримувати хороший рівень комунікації з замовниками, щоб переконатися, що існує чітке бачення кінцевого продукту і його функцій.

3. Дизайн системи. Фаза дизайну настає після того, коли досягнуто хорошого розуміння вимог споживача і точно зрозуміло, що саме треба втілити.

Ця фаза визначає елементи системи, компоненти, рівень безпеки, модулі, архітектуру, різні інтерфейси і типи даних, якими оперує система. Дизайн системи в загальних рисах може бути зроблений ручкою на листку паперу – він визначає, як система буде виглядати і як функціонувати.

Потім робиться розширений, детальний дизайн, з урахуванням всіх функціональних і технічних вимог, як логічно, так і фізично.

4. Розробка, впровадження і розгортання. Ця фаза йде після повного розуміння системних вимог і специфікацій. Це і є процесом розробки системи, коли її дизайн вже повністю завершено. В життєвому циклі розробки системи саме тут пишеться код, а також фаза впровадження може включати в себе конфігурацію і налаштування під певні вимоги і функції. На цій стадії система готова до установки у замовника, до запуску в робочий режим. Можливо, кінцевим користувачам буде потрібний тренінг, щоб вони освоїлися з системою і знали, як її використовувати.

Фаза впровадження може бути дуже довгою – це залежить від складності системи.

5. Дослідна експлуатація та інтеграція. Тут відбувається об'єднання різних компонентів і підсистем в єдину цілісну систему. В систему подаються різні вхідні дані і аналіз виходу, поведінки і функціонування.

Тестування стає все важливішим для задоволення споживача, при цьому воно не вимагає знань коду, конфігурації обладнання чи дизайну.

Тестування може виконуватися справжніми користувачами або спеціальною командою співробітників. Воно може бути систематичним і автоматизованим, з тим, щоб упевнитися, що актуальні результати роботи системи збігаються з передбаченими і бажаними

6. Підтримка системи. На цій фазі здійснюється періодична технічна підтримка системи, щоб переконатися, що вона не застаріла. Сюди входить:

• заміна старого обладнання і постійна оцінка продуктивності

• здійснюються апдейти певних компонентів, щоб упевнитися, що система відповідає потрібним стандартам і новітнім технологіям, і не схильна до загроз безпеки.

Завершуючи обговорення фази підтримки системи в рамках життєвого циклу розробки програмного забезпечення (SDLC), необхідно підкреслити, що ця стадія відіграє критичну роль у забезпеченні довготривалої ефективності та безпеки вебзастосунків. Постійні оновлення та модернізація обладнання є життєво важливими для адаптації до змін у технологічному ландшафті, що дозволяє компаніям підтримувати релевантність та конкурентоспроможність. Оскільки технології продовжують розвиватися, розробка та використання вебзастосунків стає стратегічним вибором, спрямованим на підтримку інновацій, підвищення ефективності та забезпечення успіху в динамічному бізнес-середовищі. Використовуючи вебзастосунки, компанії не тільки відкривають нові можливості для зростання, але й здатні ефективно взаємодіяти з клієнтами та адаптуватися до постійних змін у цифровому світі. Таким чином, підтримка та оновлення вебзастосунків відіграють ключову роль у використанні їх потенціалу для досягнення бізнес-цілей та забезпечення безперервної відповідності до сучасних стандартів та вимог.

Оскільки технології продовжують розвиватися, розробка та використання вебзастосунків є стратегічним вибором, який може сприяти інноваціям, ефективності та успіху в сучасному конкурентному бізнес-середовищі. Використовуючи можливості вебзастосунків, компанії можуть позиціонувати себе для зростання, ефективно взаємодіяти зі своїми клієнтами та адаптуватися до постійно мінливого цифрового середовища [9].

1.2. Аналіз загроз безпеці вебзастосунків

В сучасному цифровому світі, де вебзастосунки стають все більш інтегрованими в наше повсякденне життя та бізнес-операції, питання безпеки набуває критичного значення. Вебзастосунки, які пропонують широкий спектр послуг, від електронної комерції до соціальних мереж, стають мішенями для різноманітних кібератак. Це змушує розробників та адміністраторів вебзастосунків застосовувати комплексні методики забезпечення безпеки для захисту від атак та вразливостей. Розробка надійних механізмів захисту та застосування передових підходів у цій галузі є нагальною потребою, з огляду на постійне зростання кількості та складності кіберзагроз.

Щоб полегшити роботу фахівців з тестування безпеки, відкрита некомерційна організація з безпеки програмного забезпечення в Інтернеті OWASP (Open Web Application Security Project) досліджує та кожні кілька років оновлює список найпопулярніших уразливостей вебзастосунків [10].

На цей час найактуальнішим документом є «OWASP Top-10 2021» (до цього документ оновлювався у 2017 році). В порівнянні з минулою версією, в Top-10 2021 року було додано 3 нові категорії, 4 категорії зазнали змін в назві та в обсязі, а також було проведено деякі об’єднання. Порівняння зміни ТОП-10 вразливостей за версією організації за 2017-2020 роки можна побачити на рисунку 1.4.

Top-10 2021 року був заснований, в основному, на матеріалах, наданих компаніями, які спеціалізуються на безпеці застосунків, і на дослідженні більш ніж 500 незалежних галузевих опитувань. Ці дані охоплюють уразливі місця, отримані від сотень різних організацій та понад 100 000 реальних застосунків та API. В основі списку Top-10 лежать дані про поширеність, простоту експлуатації, а також про складність виявлення вразливостей і про можливості запобігання шкоди [11].

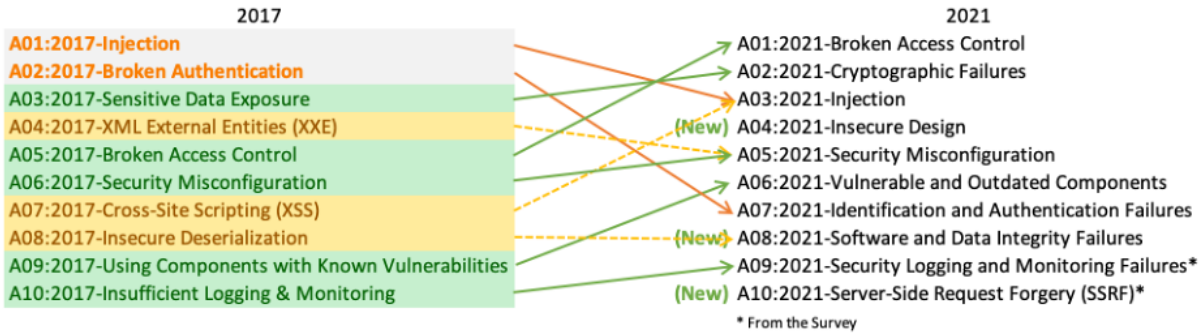


Рис. 1.4. Порівняння рейтингу вразливостей вебзастосунків за 2017 та 2020 роки [12]

За версією OWASP 2021 року до десятки найбільш популярних вразливостей для вебзастосунків входять [12]:

A01:2021 виділяє «Порушення контролю доступу» як найбільший ризик для безпеки вебзастосунків: дані показують, що 3,81% протестованих застосунків мали такі вразливості, загалом понад 318 000 екземплярів. Поширені проблеми контролю доступу включають неправильні обмеження привілеїв і обхід контролю доступу, що може призвести до несанкціонованих дій.

A02:2021 обговорює «Криптографічні збої», наголошуючи на необхідності надійного шифрування для захисту конфіденційних даних.

A03:2021 фокусується на «ін'єкціях», які часто зустрічаються в протестованих застосунках, підкреслюючи важливість ретельного тестування безпеки для різних вхідних даних застосунків.

A04:2021 представляє «Незахищений дизайн», пропагуючи проактивні заходи безпеки, такі як моделювання загроз і принципи безпечного дизайну.

A05:2021 розглядає «Неправильну конфігурацію безпеки» - поширену вразливість через погані налаштування безпеки, яка стала більш поширеною зі збільшенням кількості програмних систем, що конфігуруються.

A06:2021, «Вразливі та застарілі компоненти», звертає увагу на ризики використання компонентів з відомими вразливостями, закликаючи до регулярного оновлення та моніторингу безпеки.

Що стосується, A07:2021 - A10:2021, то вони охоплюють питання, пов'язані зі збоями автентифікації, порушеннями цілісності програмного забезпечення та

даних, неналежним моніторингом та підробкою запитів на стороні сервера (SSRF), кожна з яких вказує на критичні сфери, що потребують надійних засобів контролю безпеки та постійної уваги зі сторони адміністраторів безпеки.

Оскільки сучасні вебзастосунки надають кінцевим користувачам зручні функції, отримання URL-адреси стає поширеним сценарієм. Як наслідок, поширеність SSRF зростає. Крім того, серйозність SSRF стає вищою через хмарні сервіси та складність архітектури.

У самому документі можна знайти детальний опис кожної вразливості з зазначенням критеріїв приналежності до категорії, способи запобігання вказаних вразливостей, приклади сценаріїв атак, а також надані рекомендації щодо того, що робити далі, якщо вразливість виявлена. Отже, використання OWASP Top-10, як вважають експерти, є найефективнішим першим кроком на шляху зміни культури розробки програмного забезпечення у бік створення більш безпечного коду.

1.3. Аналіз існуючих методик виявлення вразивостей у вебзастосунках

Тестування безпеки застосунків (Application Security Testing, AST) - це процес перевірки та аналізу програми з метою виявлення потенційних вразливостей безпеки. Це не обмежується кодом застосунку, адже також включає в себе його інфраструктуру та архітектуру.

AST дозволяє передбачити і зменшити потенційні ризики безпеки, запобігти зловмисним діям і забезпечити надійність роботи застосунку. Це проактивний підхід, метою якого є виявлення вразливостей і слабких місць до того, як вони можуть бути використані. Це може бути що завгодно - від несанкціонованого доступу до ін'єкції коду, скриптових атак, перехоплення сеансів, неправильних конфігурацій і навіть помилок бізнес-логіки, які можуть створити ризики для безпеки.

Тестування безпеки застосунків - це комплексний процес, який охоплює безліч методів і методологій. Розглянемо найпоширеніші з них [13].

Тестування безпеки "чорного ящика" - це підхід, коли тестувальник не знає внутрішньої роботи програми. Цей тип тестування імітує зовнішню атаку і зазвичай проводиться з точки зору кінцевого користувача. Основною метою є виявлення вразливостей, які можуть бути використані без знання коду або архітектури. Цей метод тестує застосунок в цілому, зосереджуючись на вхідних і вихідних даних, щоб виявити проблеми безпеки, такі як помилки валідації вхідних даних, проблеми з управлінням сеансами і вразливості в зовнішніх інтеграціях.

Тестування "чорного ящика" корисне тим, що воно імітує реальні сценарії несанкціонованого втручання, де зловмисники зазвичай не мають внутрішніх знань про застосунок. Це практичний підхід для пошуку широкого спектру вразливостей, особливо тих, що впливають на вхідні дані користувача. Однак, оскільки він не заглиблюється в структуру коду, деякі вразливості можуть залишитися невиявленими. Крім того, тестування чорного ящика може зайняти багато часу і бути менш систематичним порівняно з тестуванням білого ящика, оскільки воно покладається на методи спроб і помилок.

Тестування безпеки "білого ящика", також відоме як тестування "прозорого ящика" або "скляного ящика", передбачає ретельне вивчення внутрішньої логіки та структури коду. Тестувальник має повний доступ до всього вихідного коду та архітектурної документації. Цей підхід використовується для виявлення вразливостей, які важко виявити ззовні, таких як проблеми зі шляхами коду, проблеми з виконанням коду та неправильні конфігурації безпеки. Він включає в себе такі методи, як аналіз коду, вивчення потоку даних і розуміння потоку управління.

Тестування "білого ящика" дозволяє більш повно і детально дослідити стан безпеки застосунку, оскільки воно перевіряє всі аспекти коду. Воно ефективне для виявлення прихованих вразливостей і організації безпечного кодування. Однак цей метод може бути ресурсномістким і вимагає кваліфікованих тестувальників з глибоким розумінням архітектури та мови програмування застосунку. Крім того,

незважаючи на ретельність тестування білого ящика, воно може не виявити вразливості, які стають очевидними лише під час роботи програми, наприклад, проблеми під час виконання або взаємодії з іншими системами.

Тестування безпеки "сірого ящика" - це гібридний підхід, який поєднує в собі елементи тестування "чорного ящика" та "білого ящика". Він надає тестувальнику обмежені знання про внутрішню роботу програми, як правило, доступ до деякої документації і, можливо, деякого коду. Цей підхід використовується для імітації атаки з частковими знаннями, подібними до тих, які може мати інсайдер. Тестування сірих скриньок фокусується на таких областях, як кінцеві точки API, внутрішні процеси та взаємодія між різними компонентами програми.

Перевага тестування сірим ящиком полягає в тому, що воно забезпечує більш реалістичний підхід до тестування, ніж тестування білим ящиком, оскільки не вимагає повного знання внутрішніх компонентів програми. Воно може виявити вразливості, які пропускаються тестуванням чорного ящика, але не є таким ресурсоємним, як тестування білого ящика.

Тестування сірих скриньок особливо корисне для виявлення проблем, пов'язаних з неправильним використанням API, збоями в автентифікації та управлінні сеансами, а також проблем, що виникають в результаті взаємодії між різними компонентами. Однак, як і тестування чорного ящика, воно може пропустити деякі вразливості, які глибоко вбудовані в код і можуть бути виявлені лише за допомогою ретельного підходу білого ящика.

Щодо рішень, то існує кілька типів AST. Кожне з них працює по-різному і призначене для виявлення певних типів вразливостей або для роботи на різних етапах SDLC. Розглянемо детальніше найпоширеніші з них: SAST і DAST.

DAST, або динамічне тестування безпеки застосунків, - це процес, який передбачає тестування застосунку під час його роботи. Це техніка тестування "чорного ящика", яка не вимагає доступу до вихідного коду програми [14].

Основною метою DAST є виявлення вразливостей безпеки, які можуть бути використані під час роботи програми. Він спрямований на моделювання атак з точки зору зловмисника та виявлення потенційних вразливостей, які можуть бути

використані, якщо їх не виявити. Деякі з найпоширеніших вразливостей, які може виявити DAST, включають міжсайтовий скриптинг (XSS), SQL-ін'єкції та неправильні конфігурації безпеки.

Інструменти DAST можуть тестувати будь-який застосунок, незалежно від мови програмування, що використовується. Основна увага приділяється відкритим інтерфейсам HTTP і HTML, що робить їх особливо ефективними для виявлення вразливостей, які можуть бути використані через Інтернет.

Кібератаки стають дедалі частішими та витонченішими. DAST надає реальну картину того, як застосунок може бути використаний під час його роботи. Імітуючи атаки, він виявляє вразливості, які можуть бути пропущені іншими методами тестування. Це дає розробникам чітке розуміння потенційних загроз, з якими може зіткнутися їхній застосунок, і дозволяє їм вжити проактивних заходів для їх усунення.

Крім того, DAST є мовно-діагностичним. Це означає, що його можна використовувати для тестування будь-якого застосунку, незалежно від мови програмування або фреймворку, що використовується. Це розширює сферу його застосування і робить його універсальним інструментом для забезпечення безпеки різноманітних застосунків.

DAST можна інтегрувати в конвеєр CI/CD, що дозволяє проводити безперервне тестування безпеки. Це гарантує, що проблеми з безпекою під час виконання будуть виявлені на стадії тестування та усунені до того, як вони потраплять у виробниче середовище.

Статичне тестування безпеки застосунків (SAST) - це метод тестування, який передбачає аналіз вихідного коду програми без її фактичного виконання. Це метод тестування "білого ящика", який вимагає доступу до кодової бази та мови програмування, що лежить в його основі і її розуміння.

SAST призначений для виявлення потенційних вразливостей безпеки, які можуть бути присутніми в коді. Це відбувається шляхом перевірки коду на відповідність набору попередньо визначених правил або шаблонів, пов'язаних з відомими вразливостями безпеки. Деякі з поширених проблем безпеки, які може

виявити SAST, включають переповнення буфера, небезпечні алгоритми шифрування та необроблені винятки.

SAST дозволяє виявляти вразливості на ранніх стадіях. Оскільки його можна виконати на самому початку SDLC, SAST дозволяє розробникам виявляти та виправляти проблеми безпеки ще до того, як код буде скомпільовано. Це економить час і зусилля, а також значно зменшує витрати, пов'язані з виправленням проблем безпеки пізніше.

SAST забезпечує глибокий аналіз коду. Він сканує всю кодову базу, включаючи використовувані бібліотеки та фреймворки, щоб виявити будь-які потенційні недоліки безпеки. Це забезпечує всебічне охоплення всіх аспектів програмного забезпечення, включаючи сторонні бібліотеки та пакети.

З появою DevSecOps безпека стала невід'ємною частиною життєвого циклу розробки, і SAST добре вписується в цю парадигму. Автоматизувавши сканування SAST, розробники можуть переконатися, що безпека є ключовим фактором з самого початку життєвого циклу програми.

Ці два методи тестувань мають низку ключових концептуальних відмінностей. По-перше, SAST фокусується на внутрішній структурі програми, досліджуючи вихідний код, дизайн та архітектуру для пошуку потенційних вразливостей. Це ретельний і скрупульозний процес, який може виявити широкий спектр вразливостей, від проблем з валідацією вхідних даних до переповнення буфера.

З іншого боку, DAST фокусується на зовнішній поведінці програми. Він тестує інтерфейси програми, інтерфейси HTTP і HTML, скрипти на стороні сервера тощо, щоб знайти вразливості, які можуть бути використані зловмисниками.

По-друге, SAST може виявляти вразливості, пов'язані з кодом та архітектурою програми, такі як небезпечні методи кодування, використання небезпечних бібліотек та переповнення буфера. Він також здатний виявляти вразливості на ранніх стадіях розробки, що робить його безцінним інструментом для розробників.

DAST відмінно виявляє уразливості, які видно лише під час роботи програми. Сюди входять вразливості, пов'язані з середовищем виконання, такі як небезпечні конфігурації серверів, вразливості в сторонніх компонентах і недоліки у взаємодії програми з іншими системами.

SAST забезпечує глибокий, ретельний аналіз коду та архітектури програми, але він може не охоплювати всі можливі сценарії виконання. Він чудово підходить для виявлення вразливостей у коді, але може не виявити всіх вразливостей, які можуть бути використані під час виконання.

DAST надає широкий, поверхневий аналіз поведінки програми під час виконання. Він може виявити широкий спектр вразливостей під час виконання, але не може забезпечити настільки глибокий аналіз, як SAST. Він відмінно підходить для виявлення вразливостей, які видно під час виконання, але може не виявити вразливості в коді або архітектурі.

Захист інтерфейсів прикладного програмування (API) є важливим компонентом безпеки застосунків. API, які дозволяють різним програмним системам взаємодіяти, все частіше стають мішенню зловмисників через їх доступність та чутливі дані, з якими вони часто працюють. Хоча для тестування безпеки API можна використовувати традиційні методи SAST (Static Analysis Security Testing) і DAST (Dynamic Application Security Testing), вони мають обмеження, коли справа доходить до вирішення унікальних проблем, пов'язаних з API.

SAST перевіряє вихідний код API без його запуску. SAST сканує код на наявність шаблонів, які можуть призвести до проблем з безпекою, таких як недоліки автентифікації, неправильні методи шифрування або практики кодування, які можуть зробити API вразливим до атак, таких як SQL-ін'єкція. Оскільки SAST аналізує код статично, він вміє виявляти проблеми в логіці, яка визначає поведінку API, наприклад, як він обробляє запити і відповіді, і як він обробляє дані.

DAST тестує API в запущеному стані, зосереджуючись на тому, як API поводиться, коли він працює. Цей підхід імітує зовнішні атаки, відображаючи реальні сценарії, коли зловмисники можуть використовувати вразливості, які

стають очевидними лише тоді, коли API активний. DAST оцінює відповіді API на різні типи запитів, включаючи несанкціоновані або неправильно сформовані запити, щоб перевірити наявність вразливостей, таких як витік конфіденційних даних, неправильна обробка помилок або неправильні конфігурації безпеки, включаючи управління передачею даних та автентифікацією.

Однак традиційні інструменти SAST і DAST, хоча і є цінними для певних аспектів тестування безпеки застосунків, мають обмеження, коли мова йде про комплексне вирішення проблеми безпеки API. Ці інструменти можуть не повністю охоплювати тонкощі специфічних для API вразливостей і можуть пропустити проблеми, які стають очевидними лише під час виконання. Як наслідок, зростає потреба в спеціалізованих інструментах тестування безпеки API, які можуть забезпечити спеціалізовану підтримку та глибше розуміння унікальних проблем, пов'язаних з API.

Вибір між DAST і SAST значною мірою залежить від конкретних потреб і того, на якому етапі розробки перебуває вебзастосунок. Отож, якщо застосунок перебуває на ранніх стадіях розробки, впровадження SAST буде корисним, оскільки він може виявити потенційні проблеми на рівні коду. Таке раннє виявлення дозволяє розробникам виправити проблеми до того, як вони стануть більш значними.

Якщо ж застосунок знаходиться у стані виконання, впровадження DAST може допомогти виявити вразливості, які стають очевидними лише тоді, коли програма працює. DAST також може бути корисним на завершальних етапах розробки, коли потрібна реальна перспектива того, як застосунок працює в сценаріях атак.

Однак важливо зазначити, що DAST і SAST не є взаємовиключними. Багато організацій використовують обидві технології для досягнення комплексного захисту застосунків. В той час як SAST може виявити потенційні вразливості в коді, DAST може виявити вразливості, які можна використати під час роботи програми.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ПОШУКУ ВРАЗЛИВОСТЕЙ У ВЕБЗАСТОСУНКАХ НА БАЗІ SNYK

2.1. Методика SAST

В епоху, коли безпека програмного забезпечення має першорядне значення, статичне тестування безпеки застосунків (SAST) стало критично важливим інструментом в арсеналі розробника і все більш життєво необхідним компонентом DevSecOps. Як вже було згадано раніше, завдяки ретельній перевірці програмних артефактів, SAST дозволяє командам розробників і фахівцям з безпеки виявляти вразливості на ранніх стадіях циклу розробки [15].

Статичне тестування безпеки застосунків - це методологія тестування, яка ретельно перевіряє вихідний код, байт-код або двійкові файли застосунків на наявність вразливостей без виконання програми, що лежить в їх основі.

У сучасному життєвому циклі розробки програмного забезпечення (SDLC) організація безпеки застосунку є настільки ж важливою, як і функціональність. SAST допомагає командам розробників зробити це кількома способами:

Зсув вліво: Інтегруючи SAST в SDLC, команди можуть виявляти вразливості на ранніх стадіях (так званий "зсув вліво" або "старт вліво"), що значно скорочує витрати і зусилля на виправлення на пізніх стадіях.

Ретельне обстеження: Інструменти SAST забезпечують глибоке занурення у вихідний код, виявляючи шаблони, які можуть становити загрозу безпеці.

Розширення можливостей розробників: Подібно до того, як DevSecOps зробив розробників невід'ємною частиною безпеки, SAST надає їм інструменти для створення безпечного коду з самого початку життєвого циклу вебзастосунку.

Комплексний захист: Використовуючи SAST та інші інструменти, команди можуть досягти цілісної системи безпеки, захищаючи не лише застосунок, але й усю програмну екосистему.

SAST-сканування, як правило, виконується за структурованою послідовністю для ретельного вивчення та орагнізації безпеки програмного коду. Ось огляд типових етапів SAST-сканування (рисунок 2.1):

Аналіз кодової бази: На цьому первинному етапі аналізується весь вихідний код програми, що дозволяє інструменту зрозуміти її структуру і функціональність.

Розпізнавання шаблонів: Озброєний повною базою даних відомих шаблонів вразливостей, інструмент сканує аналізований код, щоб виявити будь-які відповідні шаблони або потенційні загрози безпеці.

Визначення пріоритетів: Після виявлення потенційних вразливостей вони ранжуються або визначаються за пріоритетністю на основі їхньої серйозності, потенційного впливу та легкості, з якою вони можуть бути використані.

Цикл зворотного зв'язку: У режимі реального часу розробники отримують зворотний зв'язок про виявлені вразливості. Це забезпечує негайне вжиття заходів, що дозволяє швидше та ефективніше виправляти вразливості.

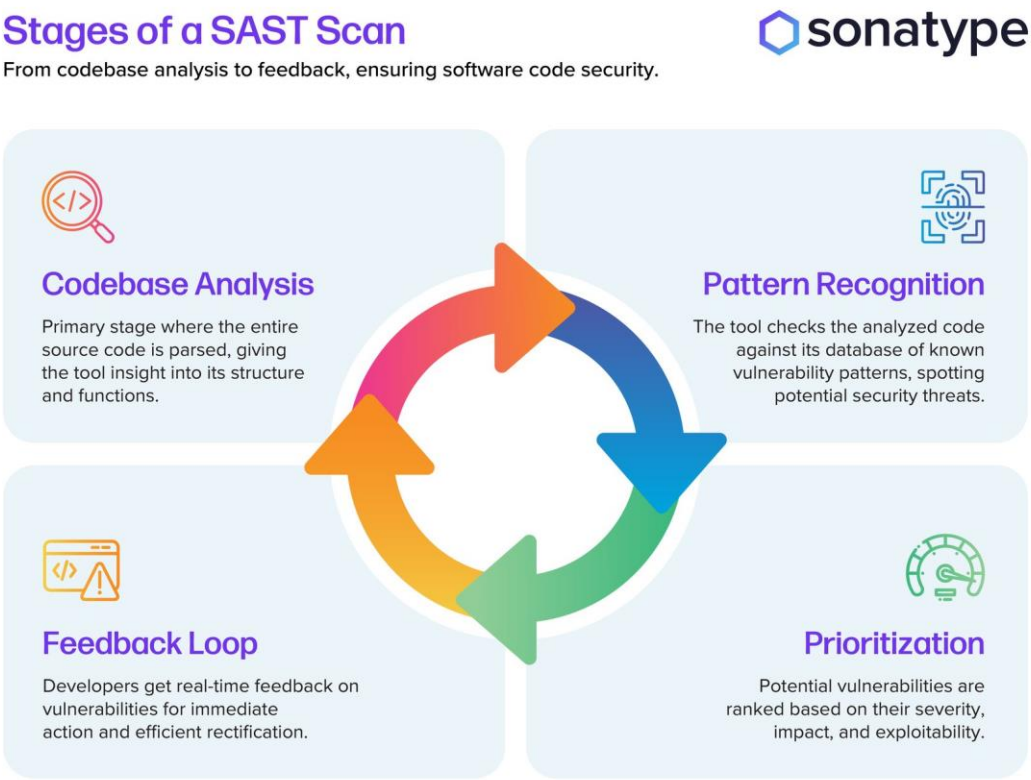


Рис. 2.1. Етапи сканування вебзастосунку за допомогою методу SAST [15]

Щодо переваг, то SAST став невід'ємним компонентом сучасної практики розробки вебзастосунків, пропонуючи численні переваги, які забезпечують більш безпечний та ефективний процес розробки. Розглянемо декілька ключових переваг цієї методології.

Проактивний захист: Однією з головних переваг SAST є його здатність виявляти вразливості ще до того, як вебзастосунок буде запущено вперше. Такий превентивний підхід означає, що потенційні загрози безпеці виявляються та усуваються заздалегідь.

Безпечне кодування: Інтегруючи інструменти SAST, організації можуть гарантувати, що їхні розробники дотримуються стандартів безпечного кодування та найкращих практик. Це не тільки підвищує рівень безпеки, але й навчає розробників нормам безпеки.

Детальні інсайти: Замість того, щоб просто позначати вразливості, інструменти SAST підтримують розробників, активно пояснюючи ризики. Вони виділяють конкретні ділянки проблемного коду і пропонують розуміння того, чому певний фрагмент може бути вразливим, а також надають пропозиції щодо виправлення.

Універсальність: Сучасні рішення SAST розроблені таким чином, щоб їх можна було адаптувати. Вони можуть працювати з різними мовами програмування, фреймворками та середовищами, забезпечуючи широку сферу застосування та сумісність.

Швидкість: У середовищі гнучкої розробки швидкість має першорядне значення. Інструменти SAST можуть надавати майже миттєвий зворотний зв'язок, гарантуючи, що процес розробки залишається швидким, без шкоди для безпеки.

Хоча SAST пропонує безліч переваг для організації безпечної розробки вебзастосунків, важливо пам'ятати, що жоден інструмент чи метод не позбавлений обмежень та недоліків. Оцінюючи роль SAST у SDLC, необхідно враховувати певні виклики та обмеження.

Однією з основних проблем, з якими стикаються при використанні статичного аналізу безпеки програмного забезпечення (SAST), є виникнення хибно

позитивних та хибно негативних результатів. Хибно позитивні результати виникають, коли інструмент помилково позначає код як вразливий, хоча насправді він безпечний. Це може призвести до зайвих зусиль з виправлення помилок, яких насправді не існує. Навпаки, хибно негативні результати виникають, коли справжні вразливості залишаються непоміченими, що може призвести до ігнорування реальних ризиків.

Ще одним обмеженням є існування контекстуальних обмежень, оскільки SAST аналізує статичний код і може не виявити вразливості, які проявляються лише в конкретному середовищі виконання або потребують взаємодії з користувачем. До того ж, обмеження динамічного середовища є значним. Деякі вразливості з'являються лише тоді, коли вебзастосунок працює, що SAST, як інструмент статичного аналізу, не може виявити. Це підкреслює важливість інтеграції інструментів динамічного тестування для більш ширшого погляду на ризики безпеки вебзастосунків.

Швидке застарівання є ще однією важливою проблемою. Світ кіберзагроз постійно розвивається, і, хоча інструменти SAST регулярно оновлюються, завжди існує затримка між появою нової загрози та здатністю інструменту її виявити. Регулярні оновлення та калібрування інструментів є життєво важливими для підтримки актуальності.

Виконання сканування SAST повинно бути інтегроване у процес розробки вебзастосунків як постійна практика, щоб забезпечити їхній високий рівень безпеки та ідентифікувати вразливості на ранньому етапі. Рекомендується проводити сканування автоматично з кожним значним оновленням коду або, принаймні, при кожному коміті в репозиторій. Це дозволить розробникам швидко виявляти та усувати потенційні вразливості перед тим, як програмний продукт перейде в наступні фази розробки або буде розгорнутий.

У випадках, коли інтеграція SAST у процес CI/CD неможлива або не практикується, рекомендується виконувати сканування щонайменше раз на етапі завершення кожного циклу розробки або спринту, залежно від використовуваної методології розробки. Додатково, проведення повторних сканувань перед запуском

виробництва або випуском значних оновлень може допомогти запобігти впровадженню нових вразливостей у продукт.

У ситуаціях, коли з'являються нові загрози або відомі вразливості, екстрене сканування SAST має бути проведене негайно, щоб оцінити потенційний вплив на існуючі проєкти та швидко реагувати на виявлені загрози.

Отже, регулярне сканування SAST є критично важливим для безпеки вебзастосунків в умовах постійних змін та загроз. Зробивши SAST-сканування послідовною та інтегрованою частиною процесу розробки, можна підтримувати надійний рівень безпеки, не заважаючи, при цьому, розвитку інновацій.

2.2. Порівняльний аналіз Snyk Code з іншими SAST рішеннями

Інструменти аналізу вихідного коду, також відомі як інструменти статичного тестування безпеки застосунків (SAST), можуть допомогти проаналізувати вихідний код або скомпільовані версії коду для пошуку вразливостей у безпеці.

Інструменти SAST можуть бути додані до інтегрованого середовища розробки (IDE). Такі інструменти можуть допомогти з виявленням проблеми під час розробки вебзастосунків. Зворотний зв'язок з інструментами SAST може заощадити час і зусилля, особливо у порівнянні з пошуком вразливостей на більш пізніх етапах циклу розробки [16].

У цьому розділі буде проведено детальний аналіз та порівняння Snyk Code з іншими провідними рішеннями у сфері статичного аналізу безпеки застосунків, зокрема, з продуктами від Veracode, Checkmarx та Synopsys. Ці інструменти були обрані для порівняння через їхнє широке визнання на ринку, передові технології аналізу та велике коло користувачів. Як можна бачити на рисунку 2.2, усі чотири рішення увійшли до лідерів магічного квадранту Ґартнера у 2023 році.

Оцінка згаданих вище рішень буде зосереджена на ключових аспектах, таких як точність аналізу, здатність виявлення вразливостей, інтеграція з

розробницькими середовищами, легкість використання, а також на підтримці мов програмування і фреймворків.

Рис. 2.2. Магічний квадрант Ґартнера для рішень класу статичного тестування безпеки застосунків [17]

Продукт SAST від Veracode забезпечує ретельний, швидкий та автоматизований зворотній зв'язок для розробників. Платформа аналізу інтегрується з популярними IDE (такими як Visual Studio, IntelliJ та Eclipse), конвеєрами CI/CD та інструментами для відстеження роботи, що робить сканування швидким і простим та надає дієві результати для розробників прямо там, де вони вже працюють. Поглиблене сканування політик перед розгортанням застосунків дає розробникам чіткі вказівки щодо пошуку, визначення пріоритетів і виправлення проблем, а керівництву і командам безпеки - загальну картину ризиків безпеки застосунків і продуктивності програм в масштабах всієї організації [18].

Checkmarx є ще одним провідним постачальником SAST-рішень, який славиться своєю здатністю до глибокого аналізу коду. Checkmarx підтримує

широкий спектр мов програмування та пропонує потужні можливості для інтеграції з різними IDE та CI/CD пайплайнами. Особливістю Checkmarx є його здатність до високої налаштовуваності правил сканування, що дозволяє користувачам тонко налаштувати процеси перевірки згідно зі специфічними вимогами безпеки [19].

Synopsys пропонує інноваційне рішення в галузі статичного аналізу безпеки програмного забезпечення (SAST), що розроблене з метою виявлення складних вразливостей у коді, які можуть уникнути уваги за стандартних підходів. Це SAST рішення відзначається не лише своєю здатністю інтегруватися безперервно в процеси розробки через CI/CD конвеєри, але й унікальною здатністю адаптуватися до складних корпоративних середовищ, що робить його ідеальним для використання в організаціях різного розміру. До того ж, рішення від Synopsys однаково швидко сканує застосунки різних розмірів - від найменших і до «найважчих».

Завдяки передовим алгоритмам глибокого аналізу, Synopsys здатний ідентифікувати тонкі вразливості та забезпечити комплексний огляд безпеки коду. Однією з ключових переваг є детальні звіти з виявленими проблемами, які супроводжуються практичними рекомендаціями для їх виправлення, що сприяє не лише покращенню безпеки проектів, але й збільшенню обізнаності та компетентності розробників у сфері кібербезпеки [20].

Snyk Code пропонує передове рішення для статичного аналізу безпеки застосунків, орієнтоване на потреби розробників. Завдяки порадам щодо виправлень, підкріпленим даними передової аналітики безпеки, затримки коду можуть бути мінімізовані, а процес розробки оптимізований. Автоматичне сканування вихідного коду прямо з інтегрованого середовища розробки (IDE) у реальному часі дозволяє отримати швидкі та точні результати, що сприяє виявленню та негайному усуненню вразливостей.

Сканування та виправлення в реальному часі елімінує необхідність очікування на звіти SAST, дозволяючи розробникам виконувати перевірку безпеки без попередньої збірки проєкту. Snyk підтримує широкий спектр мов

програмування, інструментів IDE та CI/CD, забезпечуючи комплексне покриття та адаптацію до постійно зростаючих вимог розробки.

Революційна база знань, заснована на аналізі мільйонів відкритих бібліотек та посилена можливостями машинного навчання та штучного інтелекту, забезпечує найновіші інструменти безпеки. Ця база даних дозволяє визначати пріоритети ризиків коду, акцентуючи увагу на найбільш критичних питаннях, що можуть становити значний ризик для організації.

Інтеграція з середовищем розробки значно підвищує ефективність виявлення та вирішення проблем на ранніх стадіях розробки, перш ніж вони стануть частиною основного проєкту, тим самим заощаджуючи значні обсяги часу та ресурсів. Автоматизоване сканування коду під час кожного запиту на злиття та в усіх репозиторіях дозволяє точно оцінювати та пріоритизувати безпекові проблеми, що сприяє їх своєчасному виправленню. Впровадження безпеки безпосередньо у процеси збірки через ворота безпеки CI/CD забезпечує високий рівень захисту розроблюваних продуктів, інтегруючи безпеку як фундаментальний елемент розробки [21].

Переходячи від детального огляду ключових SAST рішень на ринку до безпосереднього порівняльного аналізу, розглянемо таблицю 2.1., яка розглядає наявність критично важливого функціоналу. Ця таблиця виокремлює такі аспекти, як варіанти розгортання (on-premise, cloud, SaaS), повноту аналізу відкритих фрагментів коду (SCA), якість рекомендацій щодо усунення вразливостей, інтеграцію з розробницькими середовищами та інструментами, а також підтримку мов програмування і фреймворків. Цей аналіз дозволяє глибше зрозуміти не лише технічні характеристики кожного рішення, але й їх придатність для конкретних проєктів та організаційних потреб, висвітлюючи їхні сильні сторони та потенційні обмеження у контексті сучасних вимог до безпеки програмного забезпечення, а також ключові переваги.

Таблиця 2.1.

Порівняння рішень класу SAST

Функціонал			**Synopsys**	**Checkmarx**	**Snyk Code**	**Veracode**
1.	Варіативність розгортання (on-premise, cloud, SaaS)	on-premise	✓	✓	X	X
		cloud	✓	✓	X	✓
		SaaS	✓	X	✓	✓
2.	Повнота аналізу відкритих фрагментів коду (SCA)	SCA	✓ (окремий модуль)	✓ (окремий модуль)	✓	✓ (окремий модуль)
3.	Повнота рекомендацій з усунення виявлених вразливостей		✓	✓	✓	✓
4.	Повнота звітів з відповідності до міжнародних стандартів (compliance)		✓	✓	✓	✓
6.	Інтеграції із середовищами розробки (IDE)	Eclipse	✓	✓	✓	✓
		Visual Studio	✓	✓	✓	✓
		JetBrains	✓	✓	✓	X
		IntelliJ	✓	✓	✓	✓
		PyCharm	X	X	✓	X
		PhpStorm	✓	X	✓	X

		Jenkins	✓	✓	✓	✓
		Visual Studio	✓	✓	✓	✓
7.	Інтеграції з CI/CD інструментами	Azure DevOps	✓	✓	✓	✓
		GitHub	✓	✓	✓	✓
		GitLab	✓	✓	✓	✓
		Maven	✓	✓	✓	✓
		Terraform	X	X	✓	X
		Jira	✓	✓	✓	✓
		ServiceNow	X	✓	✓	X
8.	Інтеграції з системами управління тікетами	OpsLevel	X	X	✓	X
		Slack	✓	✓	✓	X
		GitHub	✓	✓	✓	✓
		Bitbucket	✓	✓	✓	✓
		Azure DevOps	✓	✓	✓	✓
9.	Інтеграції із джерелами отримання коду для сканування	AccuRev	✓	X	X	X
		CVS	✓	X	✓	X
		Git	✓	✓	✓	X
10.	Інтеграції із SCM	Azure	✓	✓	✓	✓
		AWS	✓	✓	✓	✓

		Google Cloud	✓	✓	✓	✓
11.	Інтеграції з провайдерами хмарних сервісів	IBM Cloud Pack	✓	X	X	X
		Amazon Elastic Container Registry	✓	✓	✓	X
		Azure Container Registry	✓	✓	✓	X
		Docker Registry	X	✓	✓	✓
12.	Інтеграції з контейнерною інфраструктурою	Google Container Registry	✓	✓	✓	X
		Nexus Repository	✓	✓	✓	X
		DigitalOcean	X	X	✓	X
		Harbor	X	X	✓	X
		JFrog	X	X	✓	✓
		Quay	X	X	✓	X

Підбиваючи підсумки, у таблиці 2.2. відображено порівняння чотирьох розглянутих вище SAST рішень за такими основними класами параметрів: точність аналізу, здатність виявлення вразливостей, інтеграція з розробницькими середовищами, легкість використання, а також підтримка мов програмування і

фреймворків. Кожен параметр оцінено по шкалі від 1 до 10, де 1 – найгірший показник, а 10 - найкращий.

Таблиця 2.2.

Загальна оцінка рішень класу SAST

Параметри	**Snyk Code**	**Checkmarx**	**Synopsys**	**Veracode**
Точність аналізу	9	8	9	8
Здатність виявлення вразливостей	9	8	8	6
Інтеграція з розробницькими середовищами	10	8	9	7
Легкість використання	10	7	6	7
Підтримка мов програмування і фреймворків	9	8	8	7
Загальна оцінка	**47**	**39**	**40**	**35**

Загальна оцінка вказує на те, що SAST рішення Snyk Code має найвищі показники за всіма параметрами, виділяючись серед конкурентів завдяки високій точності аналізу, здатності виявлення вразливостей, відмінній інтеграції з розробницькими середовищами, легкості використання та широкій підтримці мов програмування і фреймворків. Це робить Snyk Code ідеальним вибором для команд розробників, які прагнуть інтегрувати безпеку безпосередньо у свій процес розробки, підвищуючи ефективність і зменшуючи ризики безпеки. Також це робить Snyk Code рішенням-фаворитом для продовження дослідження методів та засобів пошуку вразливостей у вебзастосунках.

2.3. Інтеграція Snyk в робочі процеси IDE

Snyk - це платформа, яка дозволяє сканувати, визначати пріоритети та виправляти вразливості безпеки у коді, залежностях з відкритим вихідним кодом, образах контейнерів та інфраструктурі у вигляді конфігурацій коду [22].

Snyk забезпечує прозорість робочого процесу розробника та дає йому можливість діяти на основі отриманої інформації. Перевага полягає у залученні розробників до практик безпеки як частини їхньої роботи над розробкою. Таким чином, основна увага приділяється створенню безпечного застосунку, а не трудомісткій роботі, наприклад, встановленню жорстких воріт контролю якості.

Тепер розробники збирають застосунки з комбінації пропрієтарного та відкритого коду, запускають цей код у контейнерах, а потім розгортають інфраструктуру у вигляді конфігурацій коду, використовуючи такі технології, як Kubernetes та Terraform.

Надійний процес безпеки захищає кожен з цих компонентів там, де вони створюються та підтримуються. Snyk інтегрується в процеси DevOps, щоб працювати з розробниками, використовуючи методи, яким кожен надає перевагу, дотримуючись і підтримуючи найкращі галузеві практики. Snyk інтегрується безпосередньо у IDE, робочі процеси та конвеєри автоматизації, щоб додати експертизу з безпеки до інструментарію розробників.

Платформа безпеки розробника Snyk включає кілька ключових компонентів для забезпечення комплексного захисту програмного забезпечення на різних рівнях. Snyk Open Source призначений для ідентифікації та виправлення уразливостей у залежностях відкритого коду, тоді як Snyk Code зосереджений на виправленні уразливостей безпосередньо у вихідному коді застосунків. Для захисту контейнерів Snyk Container допомагає виявляти та виправляти уразливості в образах контейнерів та застосунках Kubernetes. Щодо інфраструктури, Snyk Infrastructure as Code (IaC) спрямований на виправлення неправильних конфігурацій у таких інструментах, як Terraform, CloudFormation, Kubernetes і

Azure, а IaC+ розширює ці можливості для виправлення конфігурацій у сервісах хмарних провайдерів, таких як AWS, Azure та Google Cloud. Детальний огляд компонентів інструментарію зображений на рисунку 2.3.

Рис. 2.3. Платформа безпеки розробника [22]

До того ж, інтеграції Snyk для процесу розробки програмного забезпечення дозволяють інтегрувати рішення у процеси розробки та безпеки, включаючи контроль вихідного коду, IDE, CI/CD та багато інших [23].

Щодо інтеграції до середовищ розробки, то плагіни та розширення Snyk Security знаходять та виправляють вразливості та проблеми безпеки у проєктах. Це дозволяє окремим розробникам, учасникам з відкритим кодом та супровідникам коду проходити перевірку безпеки, уникати дорогих виправлень на більш пізніх етапах розробки та скорочувати час на розробку та доставку безпечного коду.

Результати сканування вразливостей показують проблеми з контекстом, впливом та настановами щодо виправлення у вашій IDE, де вразливість можна виправити прямо в самій IDE [24].

Можливості інтеграцій з IDE можна реалізувати на будь-якому етапі провадження рішення у діючу систему (рис. 2.4).

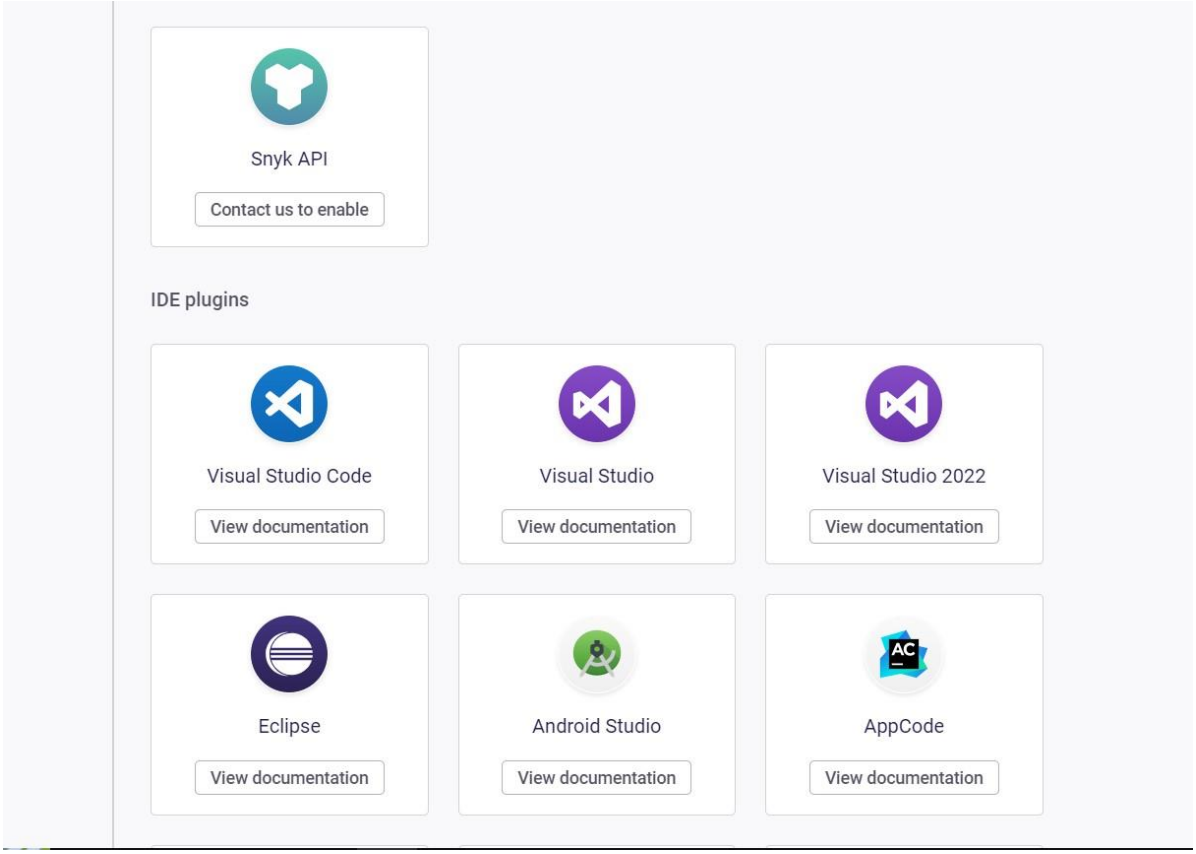


Рис. 2.4. Варіанти інтеграції Snyk з різними IDE на етапі встановлення рішення

Інтеграція Snyk з CI/CD дозволяє автоматизувати виявлення та виправлення вразливостей на різних етапах розробки програмного забезпечення. Що стосується використання інтеграції CI/CD, то зазвичай такий тип виконується поетапно. Розробник обирає метод розгортання і впроваджує стратегії для коду, який сканує. Це дозволяє командам забезпечити безперервний моніторинг безпеки їх програмного забезпечення, забезпечуючи виправлення уразливостей ще до розгортання коду в продакшн. Такий підхід підвищує загальну безпеку продукту і знижує ризики безпеки [25]. Варіанти інтеграції SAST-рішення із CI/CD можна побачити на рисунку 2.5.

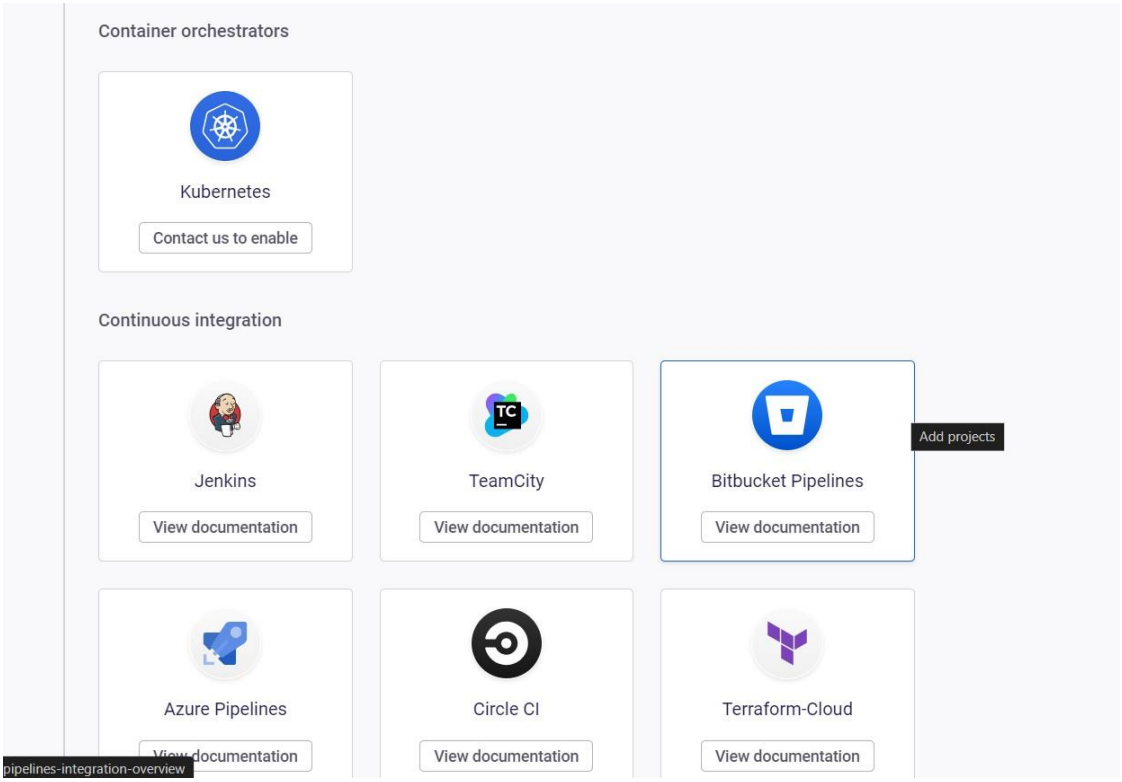


Рис. 2.5. Варіанти інтеграції рішення з CI/CD

Snyk Code підтримує широкий спектр мов програмування та фреймворків, що дозволяє розробникам інтегрувати безпеку безпосередньо в процес розробки. Підтримувані мови включають Apex, C#, C/C++, Go, Java, JavaScript, Kotlin, PHP, Python, Ruby, Scala, Swift, TypeScript та VB.NET. Це забезпечує гнучкість і охоплення для розробки безпечних застосунків у різноманітних середовищах та платформах, допомагаючи командам ефективно виявляти та усувати вразливості [26].

Важливо також зазначити, що Snyk надає спеціалізовані навчальні матеріали для розробників через платформу Snyk Learn, орієнтовані на збагачення знань про конкретні вразливості та методи їх усунення. Ці уроки корисні не лише для програмістів, але й для керівників команд та менеджерів, зацікавлених у підвищенні рівня безпеки своїх проектів та команд. Уроки у Snyk Learn відповідають категоріям робочих ролей та сферам компетенції згідно з NIST NICE Framework, забезпечуючи цілеспрямоване та систематизоване навчання [27].

Загалом, Snyk пропонує зручне та комплексне рішення для професіоналів, зацікавлених у розгортанні SAST рішення у дійсному циклі розробки застосунків у

організації. Можливості раннього виявлення, безшовна інтеграція з робочими процесами розробки, дієві рекомендації щодо виправлення, постійний моніторинг, підтримка декількох мов програмування та надійна інтеграція з CI/CD роблять Snyk ідеальним вибором для підвищення безпеки вебзастосунків [28].

3 РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ЗАСТОСУВАННЯ МЕТОДІВ ТА ЗАСОБІВ ПОШУКУ ВРАЗЛИВОСТЕЙ У ВЕБЗАСТОСУНКАХ

3.1. Варіант розгортання сканера вразливостей Snyk Code

Впровадження такого інструменту безпеки для розробників, як Snyk, має вирішальне значення для організації безпеки вебзастосунків від етапу розробки до розгортання. Незалежно від розміру команди розробників, метою рішення є усунення вразливостей і розвиток культури безпеки та відповідальності у всьому бізнес-процесі [29].

Процес розгортання рішення Snyk розпочинається з детальної оцінки поточних практик безпеки команди. Це дозволяє ідентифікувати ключові аспекти, де потрібно зосередити зусилля на покращенні. Аналізуючи існуючі стратегії, розробники можуть визначити найбільш вразливі області своїх проєктів.

Після оцінки настає етап планування, на якому формуються чіткі цілі та обираються методики їх досягнення. Важливо встановити реалістичні терміни та розрахувати необхідні ресурси, щоб забезпечити успішне впровадження платформи безпеки.

Конфігурація є ключовим моментом у адаптації платформи Snyk до специфічних потреб проєкту. Слідуючи детальним інструкціям, команди налаштовують інструментарій так, щоб він ефективно інтегрувався з існуючими робочими процесами.

На етапі розгортання команди впроваджують платформу Snyk у свої процеси розробки. Це включає навчання розробників правильному використанню інструментів для забезпечення безпеки їхнього коду. Розгортання в корпоративному середовищі може вимагати додаткових кроків для забезпечення масштабування та управління.

Навчання розробників відіграє важливу роль у процесі розгортання, оскільки підвищує обізнаність команди щодо принципів безпеки та користування платформою Snyk. Ефективне використання інструментів значно зменшує ризики для проектів.

Завершує процес регулярний моніторинг та ітерації, які допомагають відслідковувати ефективність запроваджених змін і вносити корективи. Це забезпечує постійне вдосконалення безпеки проектів у відповідності до змінюваних вимог та загроз.

Також у Snyk існує специфіка щодо розгортання в залежності від кількості команд розробників, що працюють над проєктом. Є два варіанти: розгортання для команд, що налічують менше 10 розробників (маленькі команди) та розгортання для великих підприємств, що валідне у випадках, коли SAST-рішення потрібно інтегрувати у великомасштабні процеси розробки на великих організаціях чи проєктах.

У цьому випадку буде розглянуто саме варіант розгортання для команд, що налічують менше 10 розробників. На це є ряд причин:

• по-перше, фактично, над проєктом працюватиме лише одна людина;

• по-друге, такий варіант розгортання є безкоштовним для одного розробника та включає невелику плату у випадку додавання кожного наступного працівника;

• по-третє, цей варіант є значно спрощеним, адже загальна інфраструктура проєктів не є розгалуженою, тож імплементація рішення відбувається за лічені дні;

Отож, для початку процесу розгортання необхідно перейти на офіційний вебсайт Snyk (https://snyk.io/) та зареєструватися, використовуючи один із доступних методів Single Sign-On: через GitHub чи Google-акаунт.

Після успішної реєстрації користувачеві надається вибір методу інтеграції рішення, який залежить від того, де саме опрацьовуватиметься код вебзастосунку (рисунок 3.1). До основних варіантів розгортання належать:

1) Репозиторій на GitHub
2) Butbucket Cloud
3) Інтерфейс командного рядка

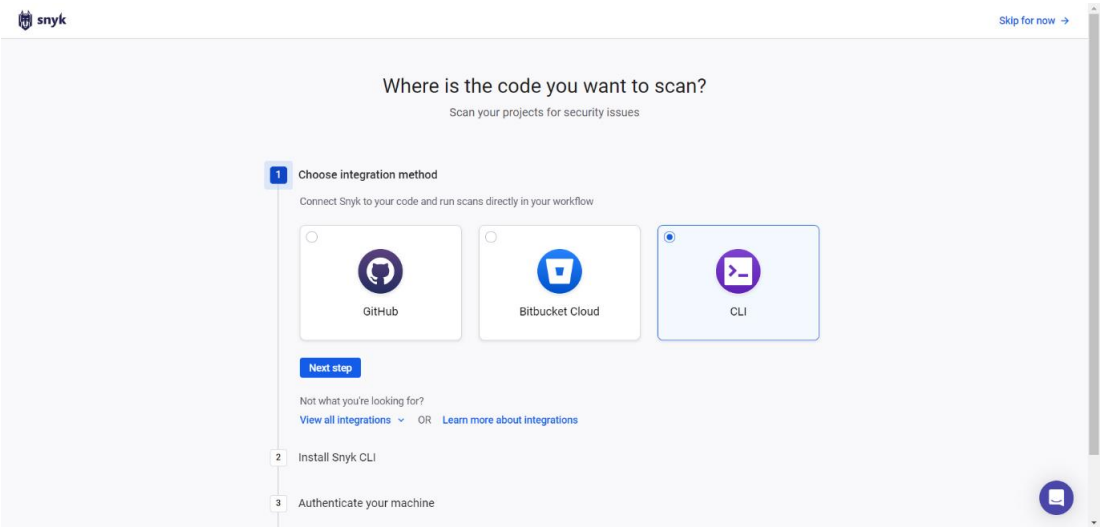


Рис. 3.1. Варіанти розгортання рішення Snyk Code

Варто також зазначити, що існують також і інші варіанти розгортань рішення [30]. Вони поділяються на класи: контроль вихідного коду (напр., GitLab, Azure Repos), реєстри контейнерів (напр., Docker Hub, Nexus, Harbor та ін.), контейнерні оркестратори (напр., Kubernetes), безперервна інтеграція (напр., Jenkins, TeamCity, Azure Pipelines та ін.), плагіни IDE (напр., PyCharm, Android Studio, Eclipse, Visual Studio Code та ін.), плагіни Gatekeeper (напр., Nexus, Artifactory Plugin) та сповіщення (напр., Jira, Slack). Приклади таких варіантів зображено на рис. 3.2.

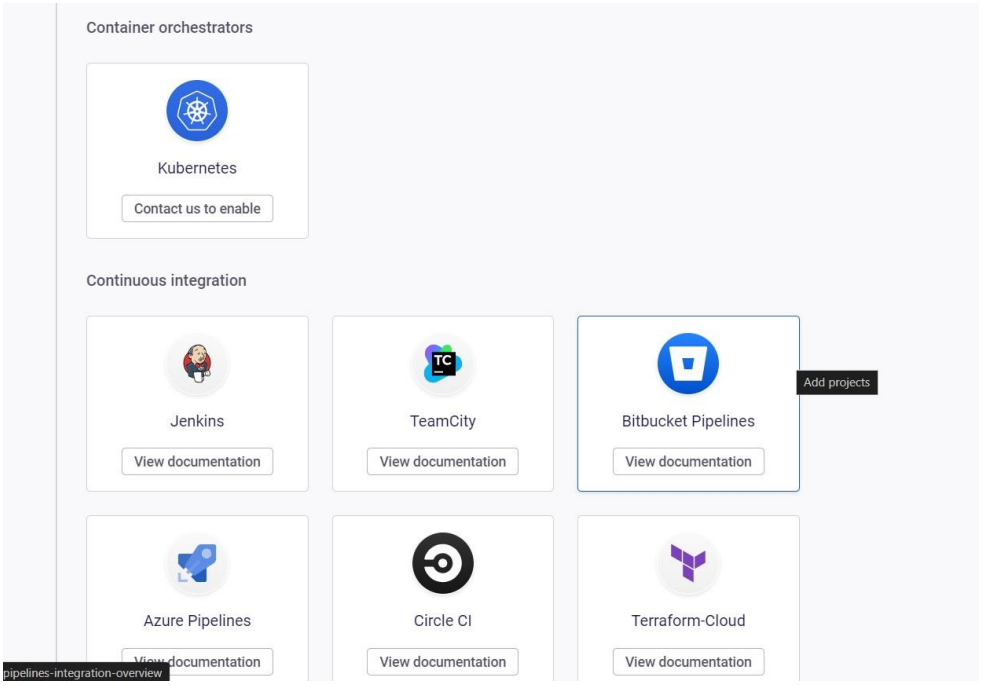


Рис. 3.2. Додаткові варіанти розгортання рішення

У цьому випадку було обрано варіант запуску через CLI, як один із найоптимальніших.

Наступний крок – вибір операційної системи, на якій буде проведено розгортання рішення. Розглянемо два найпопулярніші варіанти – Windows та Linux (рис. 3.3).

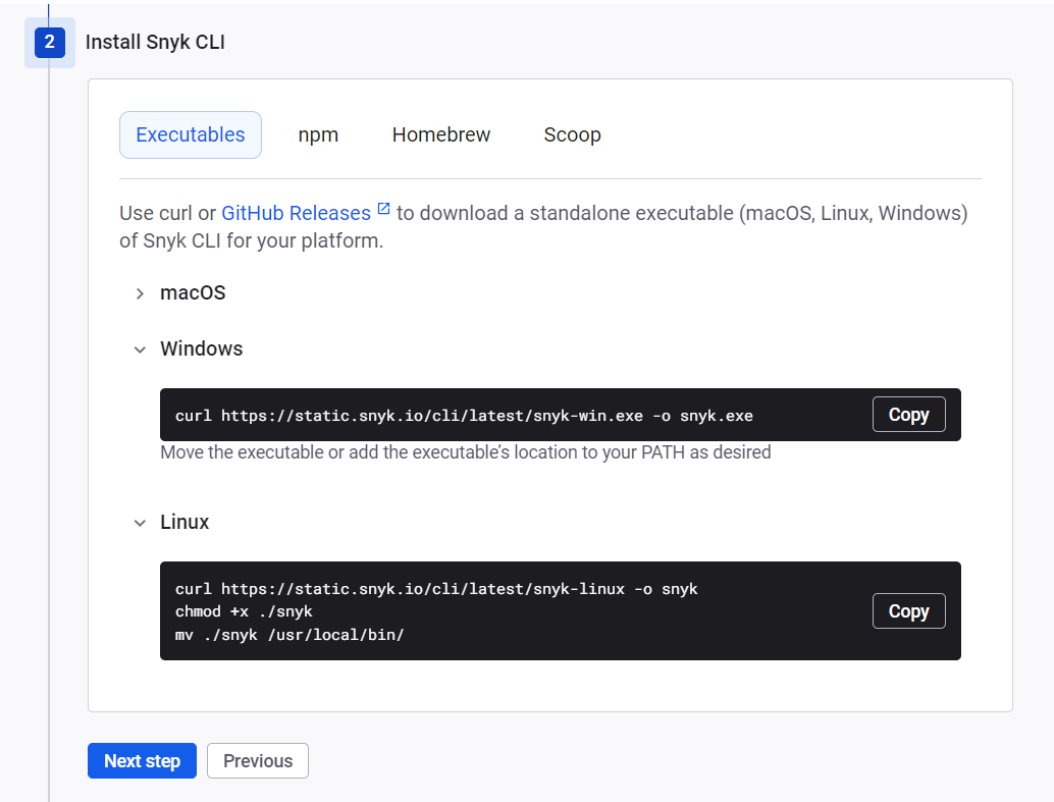


Рис. 3.3. Інструкції щодо розгортання рішення для операційних систем Windows та Linux

Як можна бачити на рисунку вище, процес розгортання є доволі тривіальним. Для ОС Windows достатньо завантажити виконуваний файл або ж за допомогою команди curl завантажити той самий файл з офіційного репозиторію. Розглянемо детальніше складові команди для завантаження та результат її виконання (рис. 3.4):

curl https://static.snyk.io/cli/latest/snyk-win.exe -o snyk.exe

• curl - утиліта командного рядка для передачі даних з або на сервер;

• https://static.snyk.io/cli/latest/snyk-win.exe - URL-адреса, з якої можна завантажити файл. Ця адреса вказує на останню версію виконуваного файлу Snyk CLI для Windows, розміщену на хостингу Snyk.

• -o snyk.exe - опція -o з наступним snyk.exe вказує curl зберегти завантажений файл з ім'ям snyk.exe у локальній файловій системі.

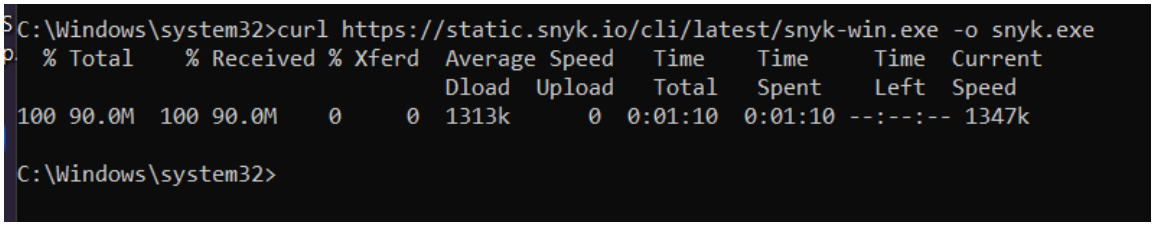


Рис. 3.4. Результат виконання інсталяційної команди

Для перевірки інстальованого рішення у системі, необхідно виконати команду «snyk» у інтерфейсі командного рядка. У разі появи інформації щодо рішення та синтаксису, SAST можна вважати встановленим у системі.

Для Linux систем алгоритм дій буде схожим, але не однаковим. Для завантаження рішення необхідно послідовно виконати команди:

curl https://static.snyk.io/cli/latest/snyk-linux -o snyk

chmod +x ./snyk

Перша команда, по аналогії із Windows, завантажує виконуваний файл у систему. Ключова відмінність цього файлу з тим, що завантажується до системи Windows полягає у тому, що у Linux для файлу потрібно надати права на виконання, використовуючи команду «chmod» та вказавши дозвіл «+x» (з англ. executable - виконуваний). Результати виконання команд та перевірку встановленого рішення зображено на рис. 3.5. Варто зазначити, що рішення розгорталося на 64-розрядній операційній системі Kali Linux.

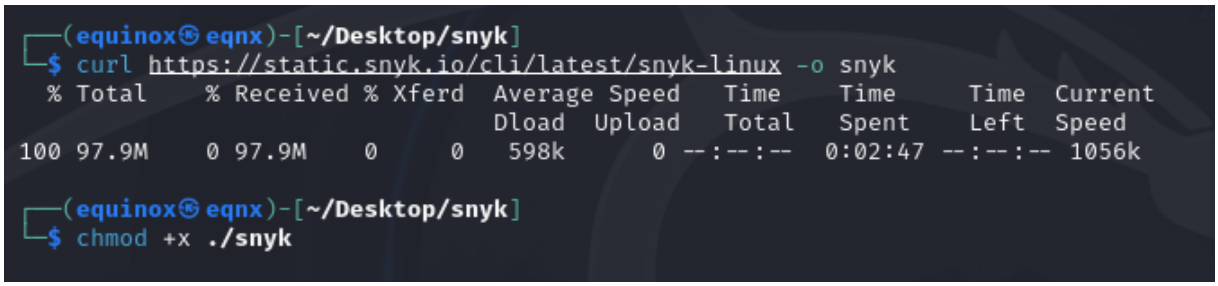


Рис. 3.5. Процес встановлення рішення на операційну систему Linux

Після встановлення Snyk, перевіримо його наявність у системі, виконавши команду «./snyk». У разі успішної інсталяції, як і у разі з операційною системою

Windows, буде відображено відомості про рішення та синтаксис із описом доступних до виконання команд (рис. 3.6).

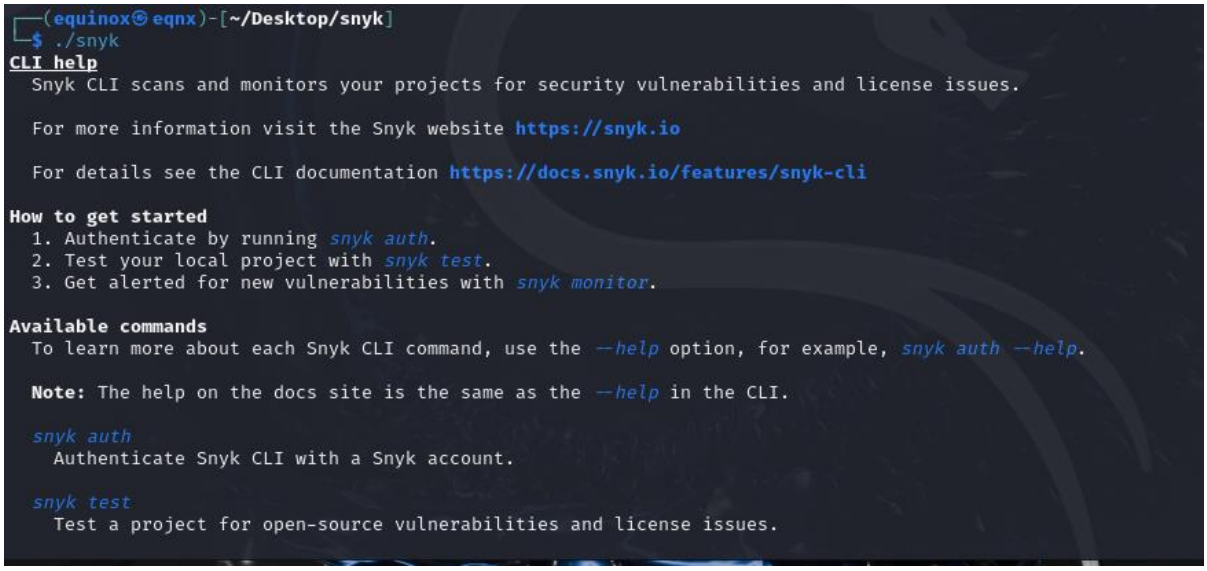


Рис. 3.6. Встановлене рішення Snyk на ОС Kali Linux

Наступним кроком інсталяції програмного продукту буде виконання автентифікації (рис. 3.7). Цей процес відкриває вікно бравзера із запитами на вхід до створеного на першому етапі облікового запису Snyk та автентифікацію комп'ютера. На цьому кроці жодних дозволів на доступ до сховища не потрібно.

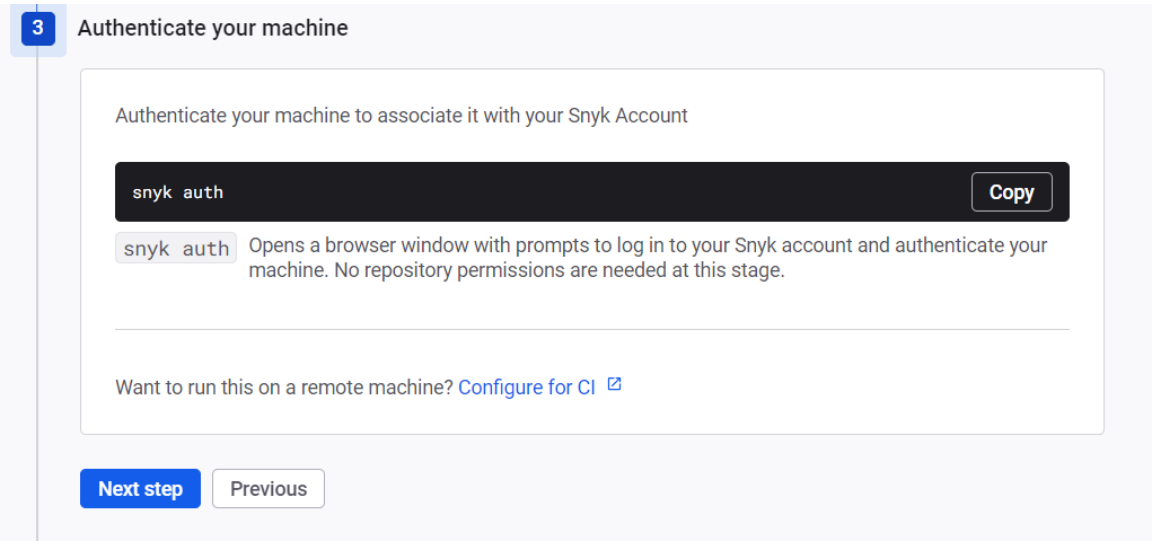


Рис. 3.7. Офіційні інструкції з проведення автентифікації облікового запису

Проведемо автентифікацію на ОС Windows. Для цього виконаємо команду «snyk auth» у середовищі командного рядку від імені адміністратора. У разі успішного виконання повинен відкритися встановлений за замовчуванням бравзер

із опцією автентифікації (рис 3.8). Далі усе, що необхідно зробити користувачеві – це слідувати інструкціям із автентифікації.

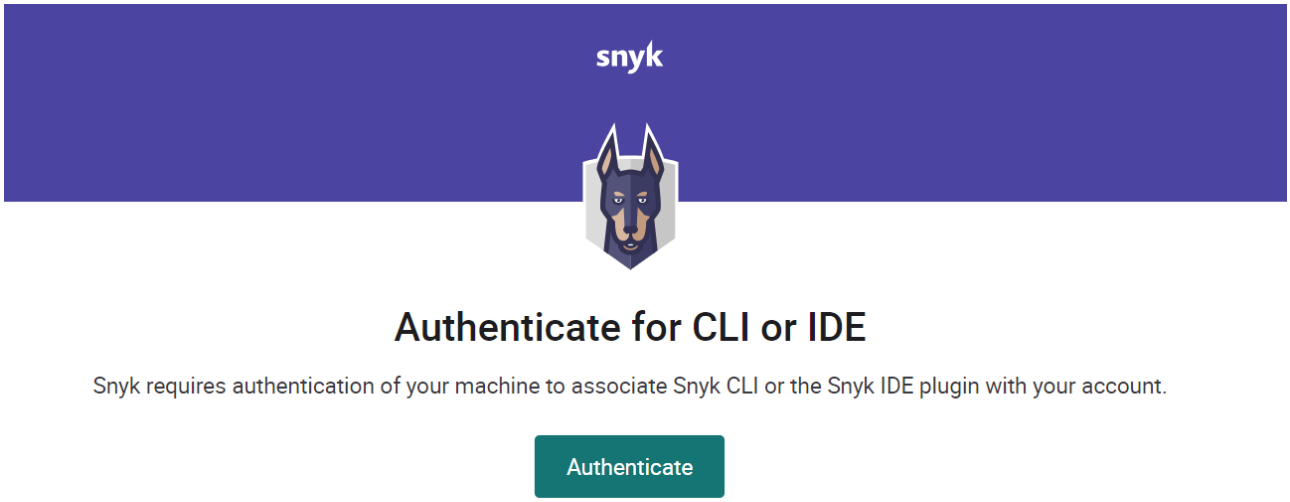


Рис. 3.8. – Вікно автентифікації Snyk

Повторимо схожу ж команду у середовищі терміналу Linux, дещо модифікуючи її:

./snyk auth

Пройшовши аналогічну процедуру автентифікації, в результаті з’явиться вікно, що повідомить користувача про успішну автентифікацію і готовність до використання (рис. 3.9).

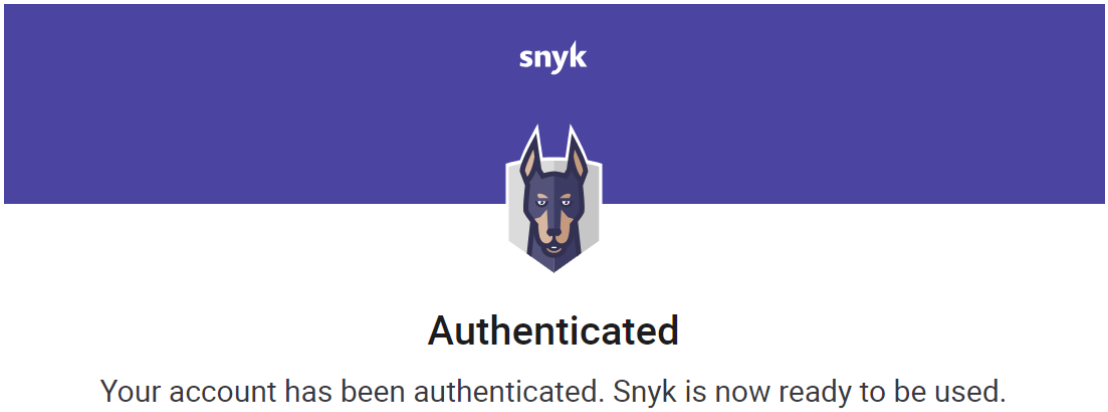


Рис. 3.9. Повідомлення про успішно пройдену автентифікацію та готовність рішення до використання

Це робить рішення готовим до наступного кроку - сканування проєктів та знаходження вразливостей у програмному коді (рис. 3.10). Цей функціонал буде детальніше розглянуто у одному з наступних розділів.

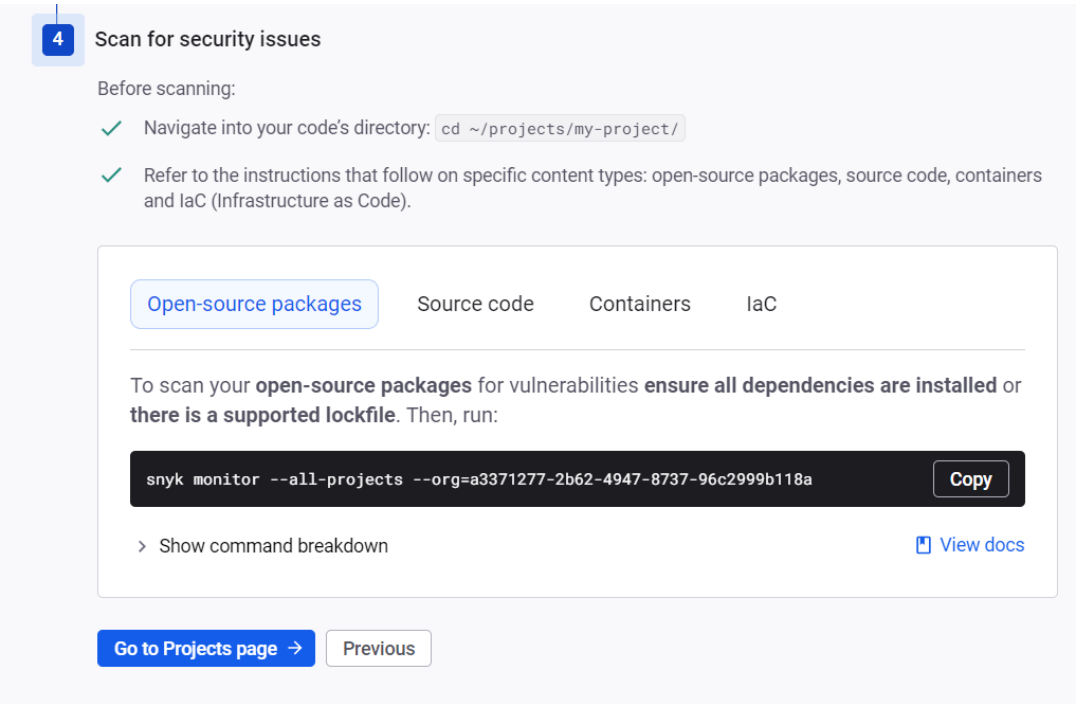


Рис. 3.10. Інформаційна довідка щодо проведення сканування коду рішенням Snyk

Також з цього моменту користувачеві стає доступним вебінтерфейс, за допомогою якого можна керувати процесом сканування, переглядати його результати та ще багато різноманітних функцій, які будуть розглянуті у наступних розділах (рис. 3.11).

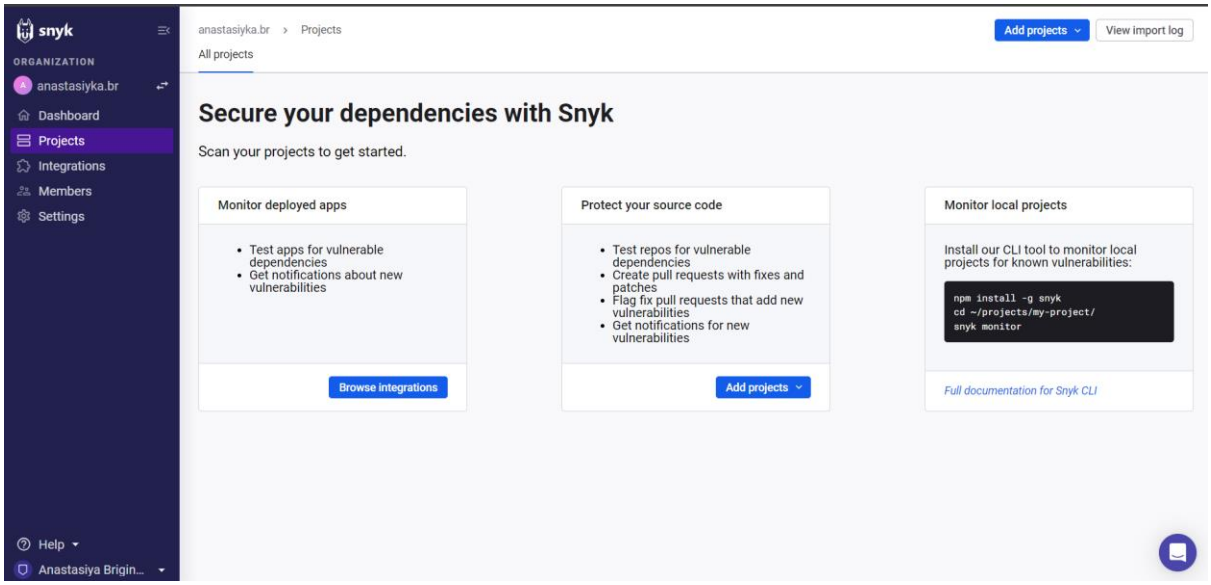


Рис. 3.11. Вебінтерфейс Snyk

Отже, впровадження інструменту безпеки Snyk є ключовим для забезпечення безпеки вебзастосунків на всіх етапах розробки і розгортання. Цей процес починається з оцінки існуючих безпекових практик, планування заходів забезпечення безпеки, налаштування інструментарію під конкретні проекти, і закінчується активним впровадженням та навчанням команд. Регулярний моніторинг та адаптація забезпечують тривале покращення безпеки, відповідаючи на змінювані умови та виклики.

3.2. Сканування коду вебзастосунків за допомогою Snyk Code

Можливості Snyk у сфері статичного тестування безпеки застосунків (SAST), аналізу складу програмного забезпечення (SCA) та аналізу інфраструктури як коду дозволяють проводити сканування кодової бази та конфігурацій хмарної інфраструктури для виявлення та усунення вразливостей. Snyk підтримує різні методи сканування, включаючи Snyk Open Source для перевірки бібліотек з відкритим вихідним кодом, Snyk Code для аналізу вихідного коду на наявність вразливостей, Snyk Container для перевірки образів контейнерів, та Snyk Infrastructure as Code для аналізу конфігурацій хмарної інфраструктури [30].

Snyk можна використовувати для ручного та автоматичного сканування коду за допомогою Snyk CLI, Snyk Web UI, Snyk API, а також за допомогою Pull Request Checks. Особливості сканування за допомогою різних типів розгортання рішення можна побачити у таблиці 3.1.

Таблиця 3.1.

Особливості сканування різних типів рішення Snyk

Особливості	Snyk Web UI	Snyk CLI	Snyk API	PR Checks
Автоматичне сканування	+	+	+	+
Ручне сканування	+	+	+	-
Локальне сканування	-	+	-	-
Включення в конвеєри CI/CD	-	+	-	-
Отримання результатів, що точно відображають вразливості та конфігурації проєкту	+	+	+	+

Найцікавішим є саме сканування за допомогою інтерфейсу командного рядка, адже саме такий тип пропонує найширший функціонал. Тому розглянемо його детальніше.

Існує набір команд, специфічних для кожного із методів тестування. У табл. 3.2 наведені ключові команди для варіанту розгортання рішення Snyk у середовищі CLI.

Таблиця 3.2.

Функціонал Snyk

Команда	**Функція**
snyk test	Сканування відкритого коду
snyk code test	Сканування коду застосунку
snyk container test	Сканування образів контейнерів
snyk iac test	Сканування файлів інфраструктури як коду (IaC)

snyk monitor and snyk container monitor	Постійне відстежування проєкту на наявність нових вразливостей.

Для наочного прикладу проаналізуємо простий вебзастосунок. У якості тестового варіанту оберемо лабораторну роботу із ґейміфікованого навчально-практичного інтернет ресурсу Hack The Box [31].

За допомогою інтерфейсу командного рядка проаналізуємо вразливий вебзастосунок під назвою “jscalc” (рис. 3.11).

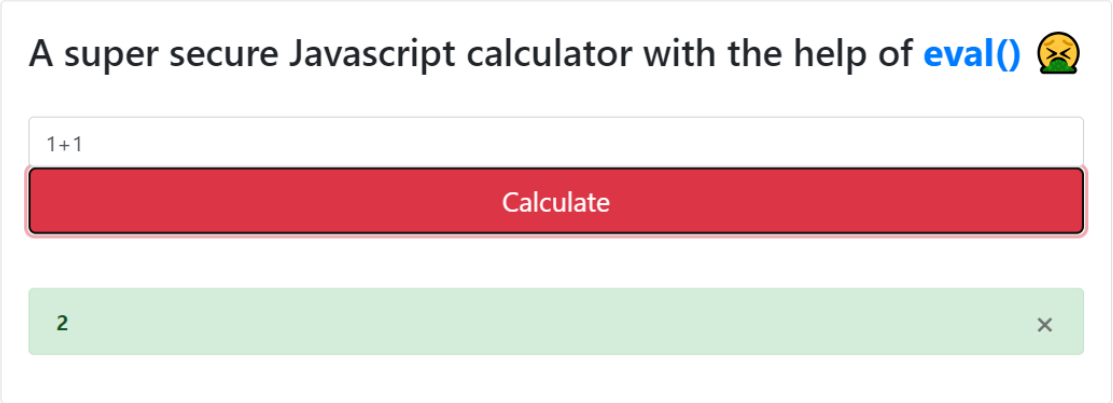


Рис. 3.11. Вид та базовий функціонал вебзастосунку

За офіційним сценарієм, «група розробників створила спеціальний калькулятор, який можна використовувати для всього - від розрахунків траєкторії струменевого літака до глибоководної математики» [32]. Перед тестувальником стоїть завдання знайти вразливість у цьому калькуляторі та де-факто виконати такі дії, які калькулятор не повинен виконувати апріорі.

Завантажимо вихідний код застосунку та спробуємо його проаналізувати. Загальна структура застосунку складається із таких файлів (рис. 3.12):

• Challenge – містить файли, що відповідають за візуальний компонент застосунку та його логіку

• Config (supervisord.config) - налаштовує Supervisord для запуску вебзастосунку Python (app.py) без демонізації, логування в стандартні потоки виводу та помилок, із вказівкою робочого директорію /app.

• build-docker.sh - автоматизує процес видалення будь-якого існуючого Docker-контейнера з іменем web_saturn, створення нового Docker-образу з тим самим іменем з поточного каталогу, а потім запуску цього образу як контейнера з іменем web_saturn на порту 1337

• Dockerfile - створює образ на основі Python 3 Alpine, встановлює необхідні залежності, налаштовує програму в каталозі /app, копіює файли конфігурації програми та Supervisord в образ, відкриває порт 1337, вимикає генерацію .pyc-файлів та налаштовує контейнер на запуск з Supervisord за допомогою наданої конфігурації.

Також існує додатковий текстовий файл - flag.txt, який є тестовим флагом для перевірки роботи застосунку та введення в оману гравців.

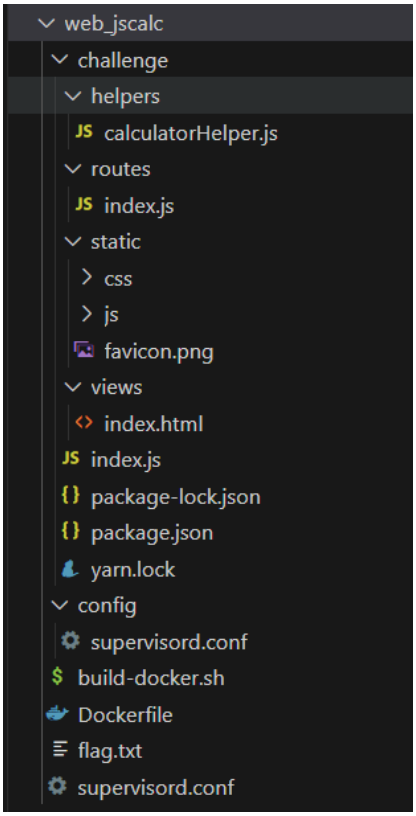


Рис. 3.12. Вихідний код вебзастосунку

Проведемо тестове сканування коду застосунку за допомогою команди «snyk test» у середовищі інтерфейсу командного рядку Windows. У результаті сканування (рис. 3.13) вебзастосунку було знайдено високу по небезпечності вразливість, відому як «Prototype Poisoning» (з англ. «отруєння прототипів»), в пакеті qs версії 6.7.0, яка була виявлена через body-parser@1.19.0 та впливає на 3 шляхи, які використовує застосунок для функціонування. На цьому етапі Snyk надає рекомендації щодо виправлення цієї проблеми: необхідно оновити body-parser до версії 1.19.2 та express до 4.17.3, оскільки обидва інструменти опосередковано використовують вразливу версію qs.

```
PS C:\Users\Lolcal\Desktop\web_saturn> snyk test

PS C:\Users\Lolcal\Desktop\web_jscalc\challenge> snyk test

Testing C:\Users\Lolcal\Desktop\web_jscalc\challenge...

Tested 52 dependencies for known issues, found 1 issue, 3 vulnerable paths.


Issues to fix by upgrading:

  Upgrade body-parser@1.19.0 to body-parser@1.19.2 to fix
  ✗ Prototype Poisoning [High Severity][https://security.snyk.io/vuln/SNYK-JS-QS-3153490] in qs@6.7.0
    introduced by body-parser@1.19.0 > qs@6.7.0 and 2 other path(s)

  Upgrade express@4.17.1 to express@4.17.3 to fix
  ✗ Prototype Poisoning [High Severity][https://security.snyk.io/vuln/SNYK-JS-QS-3153490] in qs@6.7.0
    introduced by body-parser@1.19.0 > qs@6.7.0 and 2 other path(s)



Organization:      anastasiyka.br
Package manager:   yarn
Project name:      jscalc
Open source:       no
Project path:      C:\Users\Lolcal\Desktop\web_jscalc\challenge
Licenses:          enabled
```

Рис. 3.13. Результат сканування вебзастосунку

Така інформація про інцидент є необхідною, але не зовсім достатньою. Тому для подальшого виправлення вразливості рекомендується додати інцидент до вебінтерфейсу Snyk для більш наочного аналізу:

snyk monitor --all-projects --org=a3371277[redacted]118a

Результатом виконання цієї команди буде додання інформації про вразливість до вкладки «Проєкти» у вебінтерфейсі Snyk.

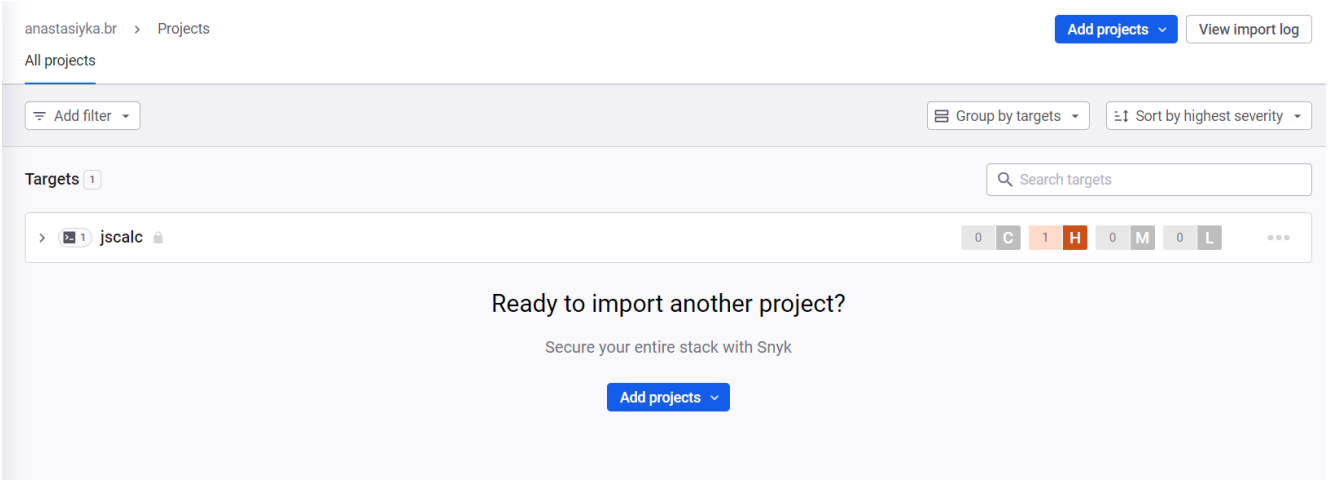


Рис. 3.14. Додана інформація про сканування вразливості вебзастосунку

Розділ детального огляду вразливості (рис. 3.15.) представляє інформацію про записи у базах CWE, CVE, оцінку за CVSS та записи у базі даних Snyk. Тут оцінюється рівень критичності знахідки а також надається змога прочитати офіційну документацію від Snyk щодо анатомії можливої атаки, векторів її реалізації та шляхів усунення.

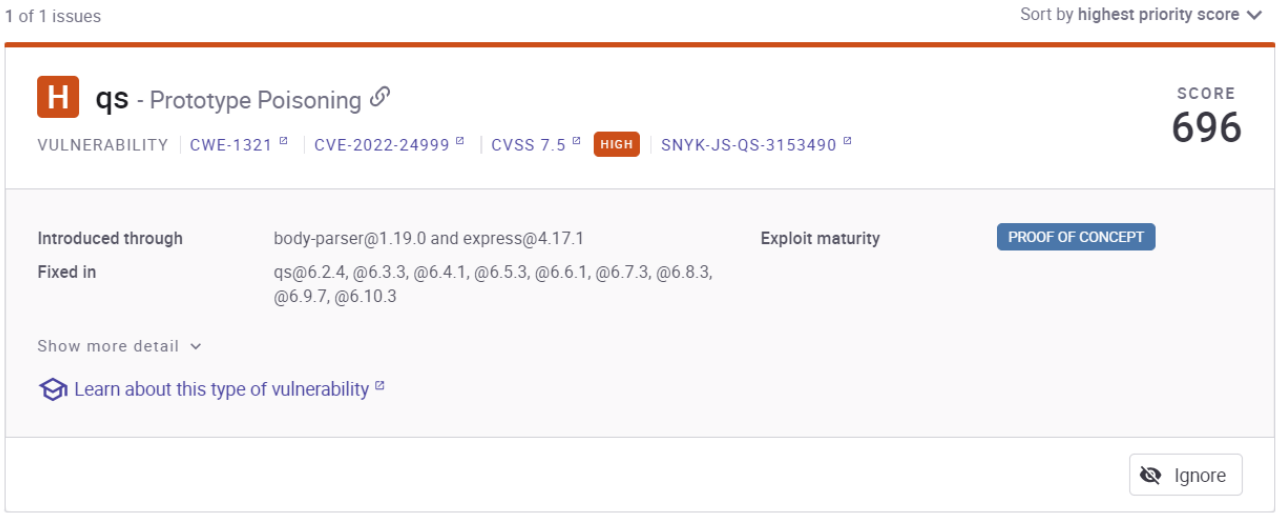


Рис. 3.15. Детальний огляд вразливості

Звідси ми можемо взнати, що забруднення прототипів - це ін'єкційна атака, яка націлена на середовище виконання JavaScript. За допомогою забруднення прототипів зловмисник може контролювати значення властивостей об'єкта за замовчуванням. Це дозволяє зловмиснику втручатися в логіку роботи програми, а

також може призвести до відмови в обслуговуванні або, в крайньому випадку, до віддаленого виконання коду (анг. Remote Code Execution (RCE)).

Однак виявлення вразливих версій проміжного програмного забезпечення не дуже схоже на специфіку роботи рішення класу SAST, скоріше всього, це спрацював аналіз складу програмного забезпечення (з англ., SCA - Software composition analysis). Тому необхідно специфікувати команду саме на активацію сканування Snyk Code – SAST рішення:

snyk code test

Результат сканування показав, що код містить вразливість ін’єкції шкідливого коду. Завдяки вбудованій у плагін довідці, користувач може дізнатися, що така атака відбувається, коли зловмисник використовує існуючу вразливість обробки вхідних даних, передаючи спеціальні символи та код безпосередньо до вебзастосунку або сайту. Потім код виконується, потенційно надаючи користувацькій системі доступ до експорту конфіденційних даних, інсталяції шкідливого програмного забезпечення або навіть до інших систем у довіреному внутрішньому мережевому середовищі. Хоча атаки на впровадження коду можуть відбуватися кількома способами, їхньою спільною рисою є надання користувачеві можливості надсилати виконуваний код до програми. Найпоширенішими формами ін'єкції коду є SQL-ін'єкція (на стороні сервера) та міжсайтовий скриптинг (XSS) (на стороні клієнта).

За оцінкою Snyk Code, ця вразливість має рейтинг загрози «Високий», тож саме ця загроза, найбільш ймовірно і буде проексплуатована потенційним зловмисником.

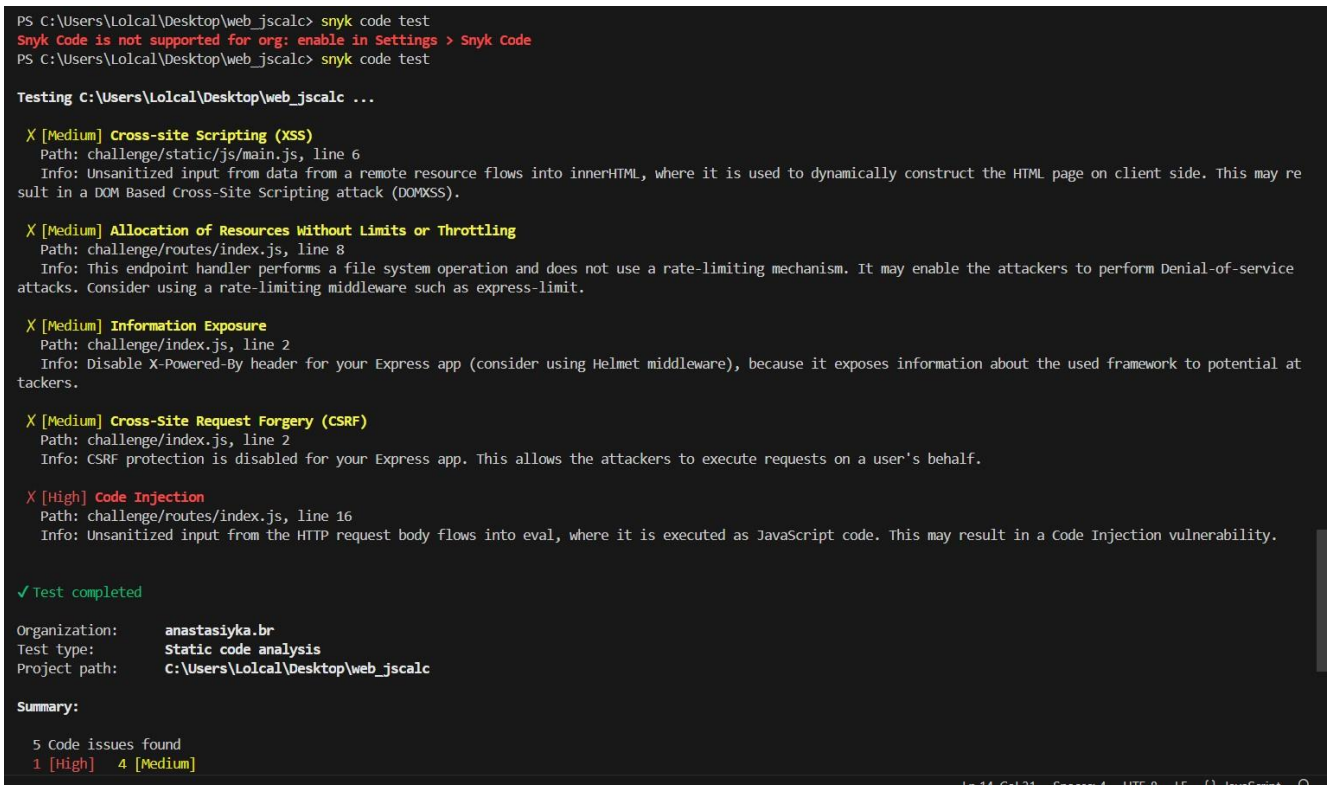


Рис. 3.16. Результат сканування Snyk Code

Перевіримо процес експлуатації такої вразливості. Для цього на основній сторінці вебзастосунку напишемо будь-який математичний приклад, надішлемо запит на його обробку, який, в свою чергу, перехопимо за допомогою Burp Suite (рис. 3.16).

Рис. 3.16. Перехоплений POST запит за допомогою Burp Suite

Як можна бачити, калькулятор використовує деяку функцію «formula», яку можна було б змінити так, щоб викликати віддалене виконання коду (RCE), маніпулюючи логікою вебзастосунку для отримання флагу, тобто успішної експлуатації вразливості.

Отже, маніпулюючи функцією, ми можемо заставити її прочитати потрібний нам файл, задавши такі значення:

{"formula":"require('fs').readFileSync('/flag.txt').toString();"}

Результатом виконання оновленого запиту буде вивід вмісту файлу flag.txt та успішна експлуатація вразливості (рис. 3.17)

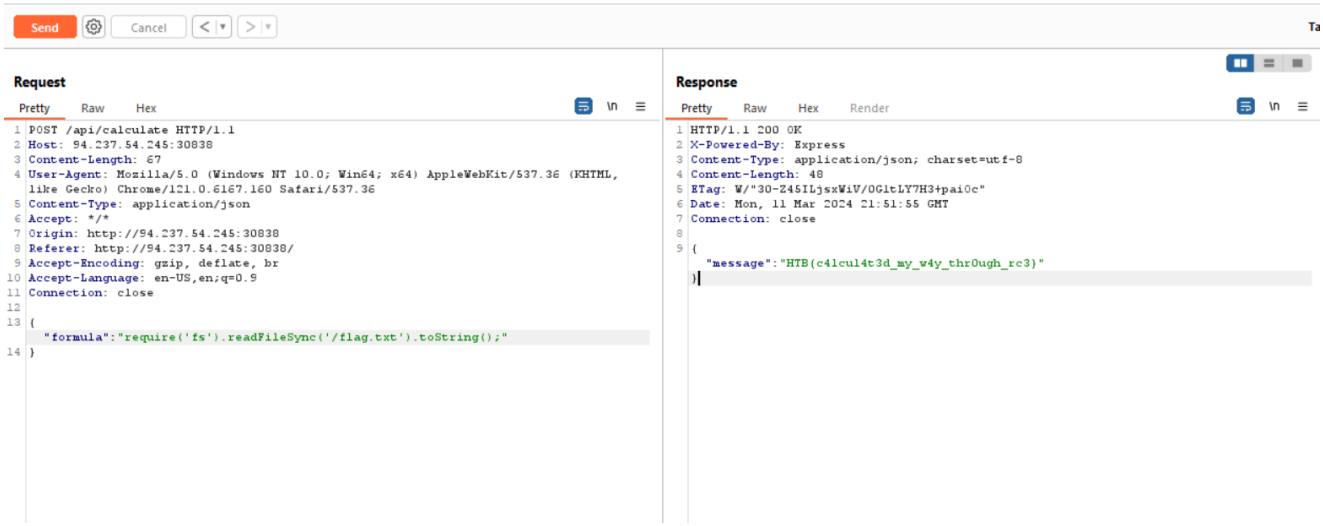


Рис. 3.17. Успішна реалізація вразливості

Щодо усунення проблеми, то Snyk свідчить, що завжди необхідно встановлювати фільтри на введення даних при рекурсивному встановленні вкладених властивостей. Однак робити це вручну не рекомендується: замість цього можна використовувати бібліотеку, таку як lodash, яка є надзвичайно популярною і має чудову підтримку спільноти та досвід швидкого виправлення проблем безпеки [33].

Також Snyk рекомендує оновити проміжне програмне забезпечення express та body-parser до новіших версій, у яких така вразливість уже виправлена (рис. 3.18).

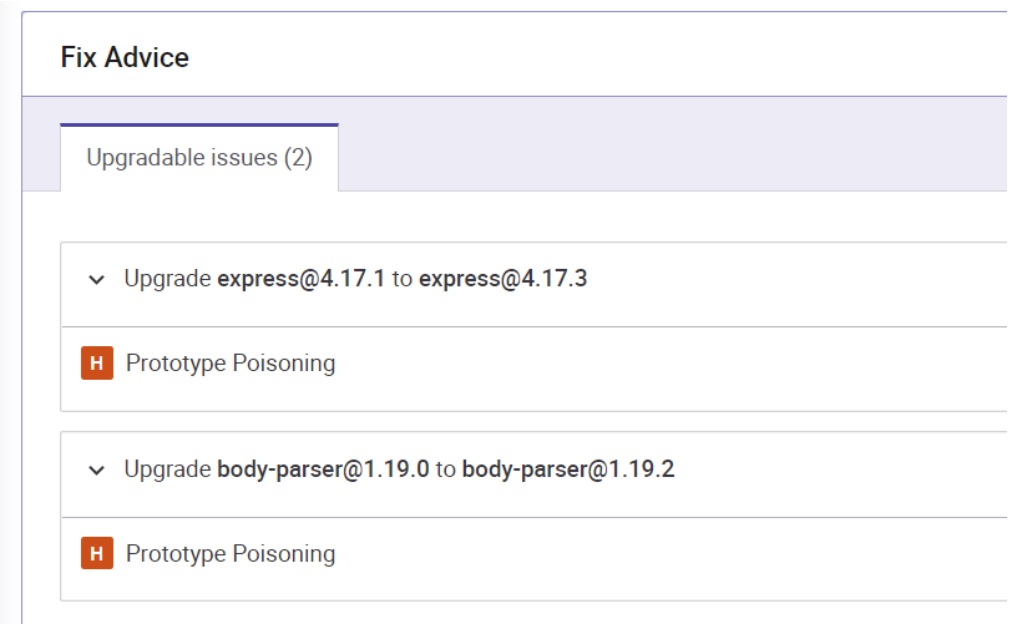


Рис. 3.18. Рекомендації щодо виправлення вразливості

Отже, впровадження інструменту Snyk значно підвищує захищеність вебзастосунків завдяки використанню комплексних підходів статичного тестування безпеки програмного коду. Це дозволяє виявляти вразливості на різних етапах розробки і застосування, пропонуючи ефективні методики їх усунення та попередження. Розширення можливостей Snyk через інтеграцію з різними середовищами та платформами забезпечує адаптацію до специфічних потреб проектів і розвитку культури безпеки в командах розробників.

3.3. Рекомендації щодо інтеграції Snyk в розробку програмного забезпечення

У розділі 2.3. було розглянуто ключові інтеграції Snyk, які спрощують процеси розробки та забезпечення безпеки програмного забезпечення, а також підтримку широкого спектра мов програмування та фреймворків. Настав час зосередитися на практичній складовій, зокрема, на інтеграції Snyk у середовище розробки Visual Studio Code. За даними [34], маючи майже 75% частки ринку та 14 мільйонів користувачів, VSCode є найпопулярнішою IDE. Враховуючи такі

фактори, це середовище розробки є ідеальною платформою для впровадження інструментів Snyk, що дозволяє ефективно організовувати безпеку програмного коду на кожному етапі розробки.

У розділі інтеграцій бравзерного інтерфейсу Snyk оберемо плагін для VSCode (рис. 3.20).

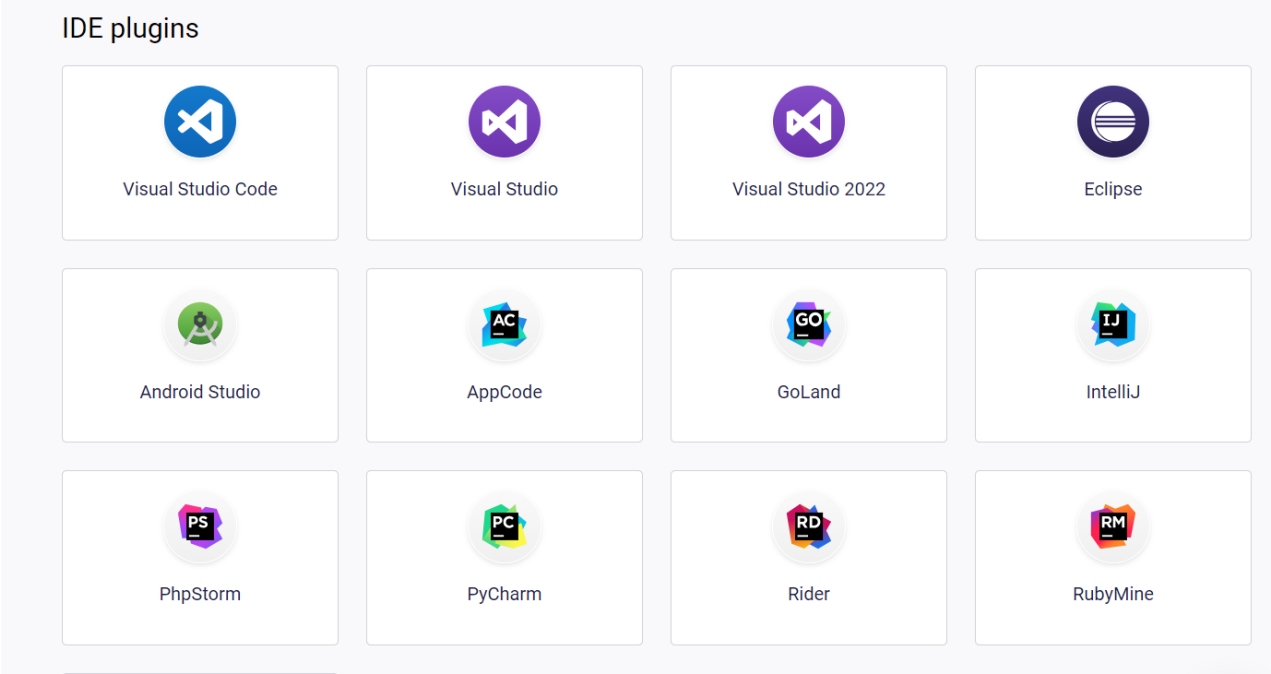


Рис. 3.20. Можливості інтеграцій Snyk з різними IDE

Перейшовши за посиланням, відкривається сайт з плагіном Snyk для IDE (рис. 3.21).

Як зазначає розробник, у плагіні на основі алгоритмів для відкритого коду надаються автоматизовані пропозиції для виправлення як прямих, так і транзитивних залежностей. Результати відображаються у контексті в інтегрованому середовищі розробки, забезпечуючи безпосередній доступ до інструментів виявлення вразливостей. Використання розширення Visual Studio Code дозволяє користуватися базою даних вразливостей Snyk та Snyk Code AI Engine [35].

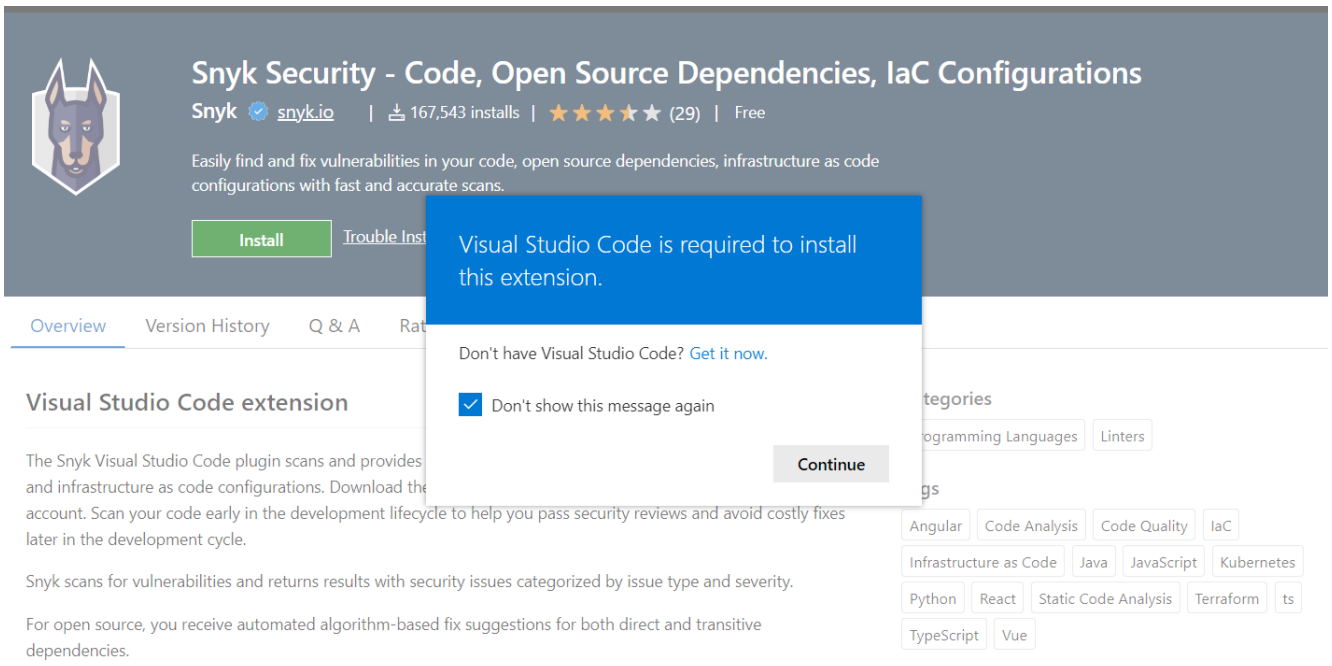


Рис. 3.21. Встановлення плагіну з офіційного сайту

Після успішної інсталяції плагіну, можна перевірити його наявність у меню плагінів самої IDE. Повинно значитись, що плагін завантажено.

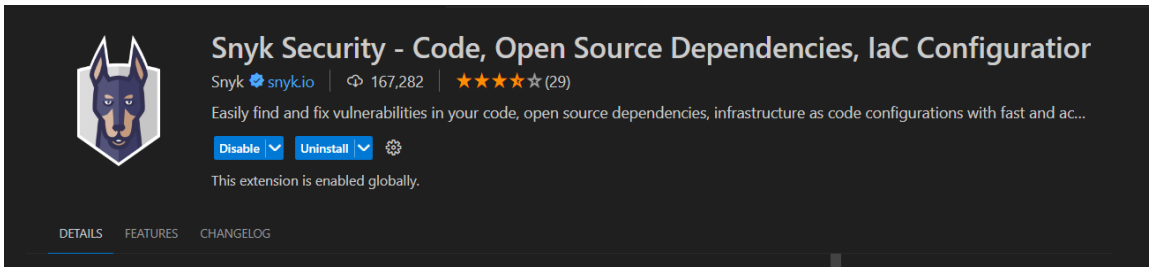


Рис. 3.22. Завантажений плагін у середовищі VSCode

Після встановлення плагіна можна встановити наступні конфігурації для розширення (рис. 3.23):

• Токен: ввід токену, який плагін використовує для підключення до вебконсолі Snyk. Можна замінити його вручну, якщо потрібно перейти на інший обліковий запис.

• Індивідуальна кінцева точка: можна вказати індивідуальну кінцеву точку API Snyk для організації. Це поле також використовується для налаштувань Single Tenant замість https://app.snyk.io.

• Ігнорування невідомого центру сертифікації: це налаштування ігноруватиме невідомі центри сертифікації.

• Організація: В цьому налаштуванні можна вказати ORG_ID для запуску команд Snyk, прив'язаних до певної організації. Snyk рекомендує використовувати ORG_ID. Якщо вказувати ORG_NAME, тобто ім'я мітки організації, то це значення має збігатися з міткою URL, яка відображається в URL організації в інтерфейсі Snyk: https://app.snyk.io/org/[orgslugname]. Якщо не вказано, для запуску тестів використовується організація, якій надається перевага (як визначено в налаштуваннях облікового запису).

• Надсилання аналітики використання: щоб допомогти Snyk покращити розширення, можна надати Visual Studio доступ до надсилання Snyk інформації про те, як працює розширення.

• Налаштування проєкту: будь-які додаткові параметри Snyk CLI. Наприклад, для всіх .NET проєктів Snyk рекомендує додати додатковий параметр --all-projects.

• Сканувати всі проєкти: автоматичне виявлення всіх проєктів у робочому каталозі (увімкнено за замовчуванням).

• Налаштування виконуваних файлів: відмова від завантаження CLI через плагін і використання власної інсталяції CLI.

Якщо позначено пункт «Автоматично керувати потрібними двійковими файлами», то плагін автоматично завантажує CLI і оновлює його.

Якщо пункт «Автоматично керувати потрібними двійковими файлами» знято, необхідно вказати правильний шлях до CLI. Цю опцію необхідно використовувати, якщо завантаження CLI неможливе через конфігурацію мережі (наприклад, через правила міжмережевого екрану) і потрібно отримати CLI іншими способами. Snyk рекомендує завжди використовувати найновішу версію CLI [36].

• Налаштування рішення: встановлення додаткових параметрів snyk test CLI для сканування відкритого коду. Для некерованого сканування на C/C++ необхідно використовувати параметр CLI --unmanaged для пошуку уразливостей у пакетах з відкритим кодом. Для цього потрібно вимкнути опцію «Сканувати всі проєкти». Параметр --unmanaged працює лише для некерованого сканування на C/C++; його

використання для інших мов забороняється. Додаткові параметри не застосовуються до Snyk Code або IaC.

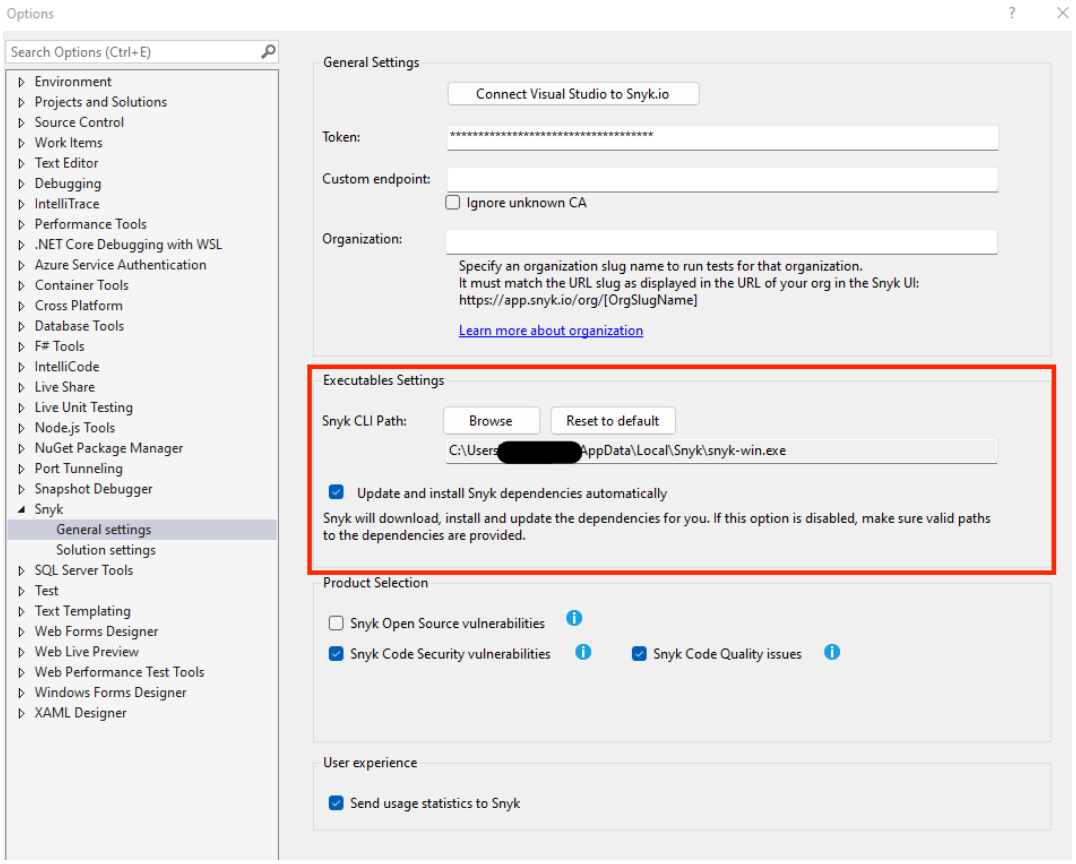


Рис. 3.23. Налаштування плагіну [41]

Після того, як усі необхідні налаштування було виконано, можна переходити до тестової перевірки функціоналу інтегрованого рішення. Для експерименту було обрано вебзастосунок з попереднього розділу.

Відразу після встановлення плагіну та завершення налаштування, почалась робота по виявленню помилок у програмному коді. Snyk аналізує відразу декілька напрямків:

1) Open Source Security
2) Code Security (безпосередньо SAST)
3) Конфігураційні проблеми

Як показано на рис. 3.24, статичний аналіз виявив ті ж самі вразливості, що і було показано у попередньому розділі, однак в конкретній реалізації рішення показує точне місце знаходження вразливості – рядки з програмним кодом.

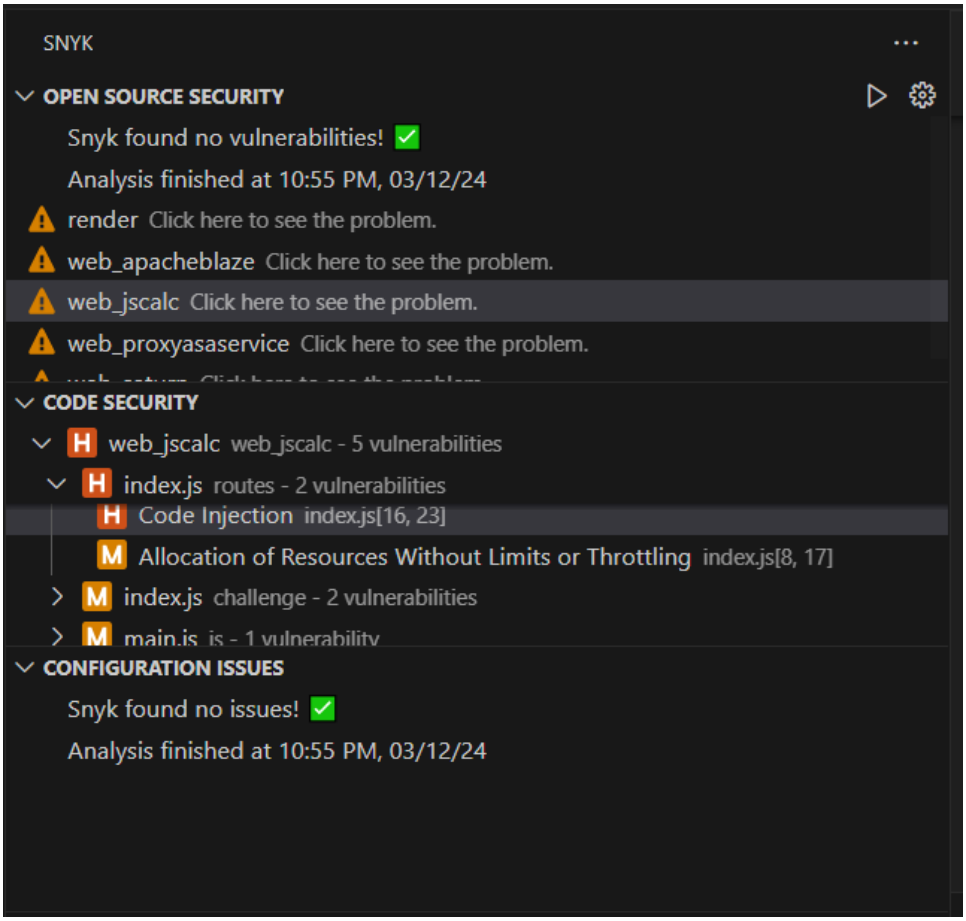


Рис. 3.24. Виявлення вразливостей з інтегрованим у VSCode рішенням

Додатково можна переглянути коротку інформацію про саму вразливість безпосередньо у середовищі розробки (рис. 3.25). Серед доступної інформації: оцінка критичності вразливості, її класифікація, гіпотетична пріоритетність та деталі реалізації.

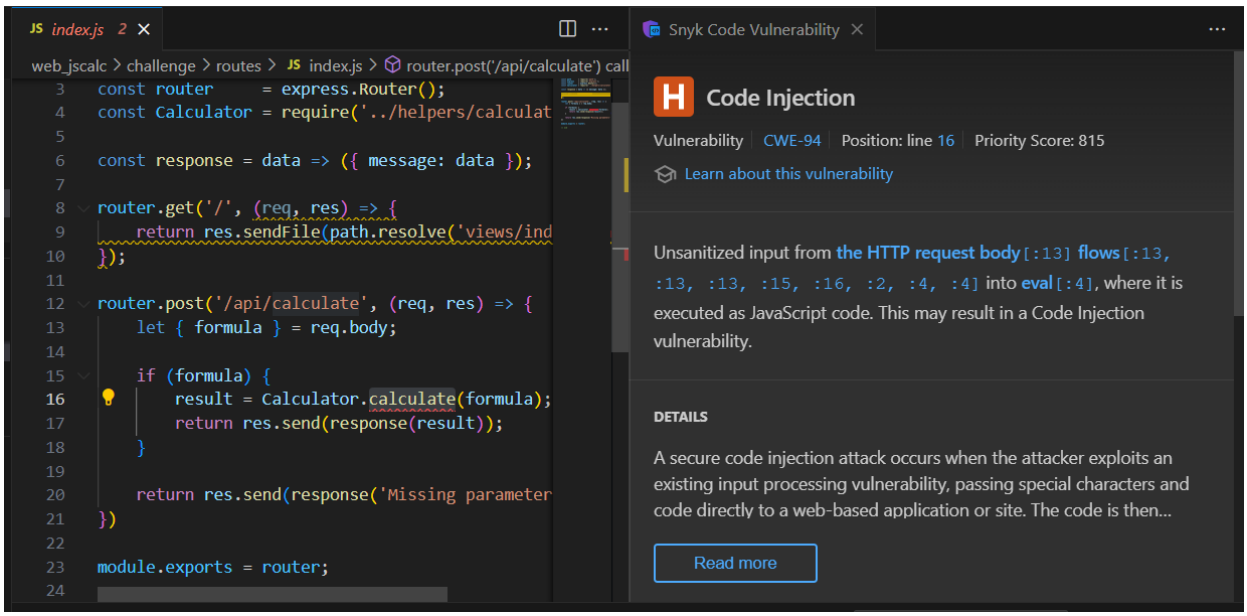


Рис. 3.25. Додаткові відомості про знайдену вразливість

Отже, використання Snyk у VSCode не тільки ефективізує процес ідентифікації та виправлення вразливостей, але й виховує культуру безпеки серед розробників, спонукаючи їх до більш усвідомленого підходу до безпеки коду. Такий інтегрований підхід сприяє створенню більш безпечних програмних продуктів і зменшенню потенційних ризиків для кінцевих користувачів, підвищуючи довіру до розроблених рішень.

3.4. Моніторинг та звітність про безпеку програмного забезпечення

Snyk Reports надає платформу для аналізу вразливостей безпеки, доступну через спеціальну вкладку. Різні звіти можна переглядати за допомогою випадаючого меню з можливістю фільтрувати інформацію на основі декількох атрибутів, налаштовуючи аналіз під конкретні потреби. Незастосовані фільтри залишаються неактивними. Платформа використовує URL-адреси зі збереженням стану подання із застосованими фільтрами, що дозволяє легко ділитися цими

відфільтрованими даними з авторизованими членами організації або групи Snyk, посилюючи співпрацю з питань безпеки.

Для безперервного відстеження проєктів та перегляду останніх снепшотів у вебінтерфейсі з метою покращення ведення звітності можна використовувати команду, результат якої відображено на рис. 3.36:

snyk monitor --all-projects --org=a3371277-2b62-4947-8737-96c2999b118a

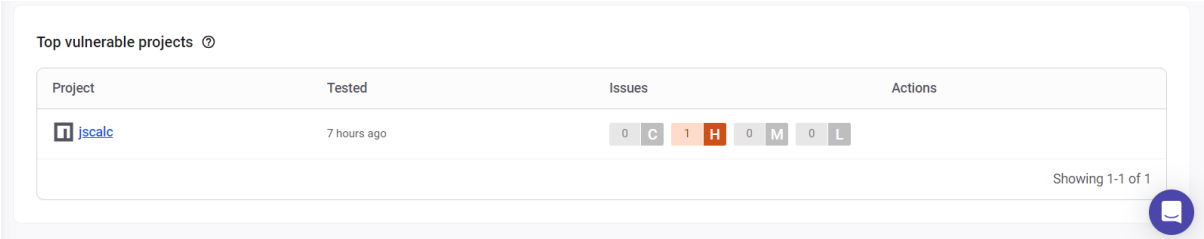


Рис. 3.36. Результати сканувань проєктів на головній сторінці «Dashboard» (рейтинг найвразливіших проєктів)

Звітність по всім проєктам можна побачити на вкладці «Projects» (рис. 3.37).

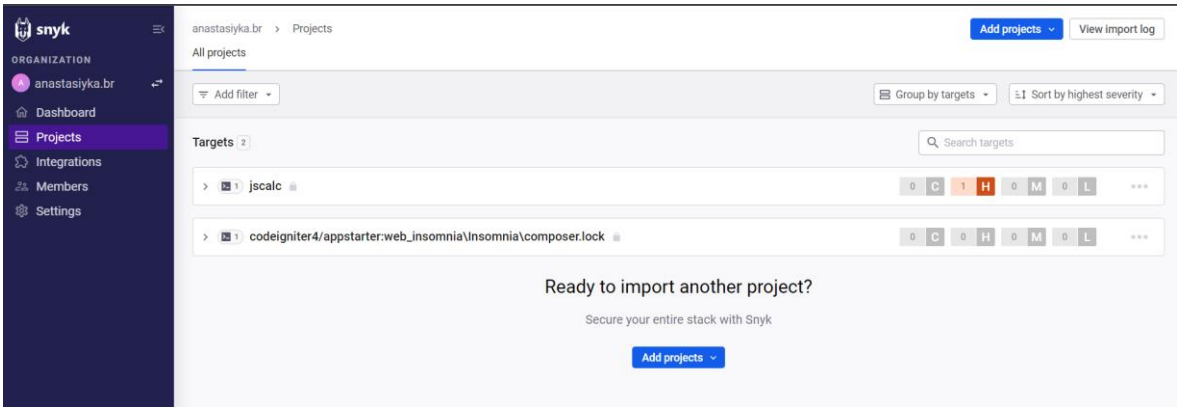


Рис. 3.37. Усі проєкти

Також у Snyk є функціонал створення та перегляду окремої звітності по кожному з проєктів, за умови вибору відповідного функціоналу у меню проєкту (рис. 3.38).

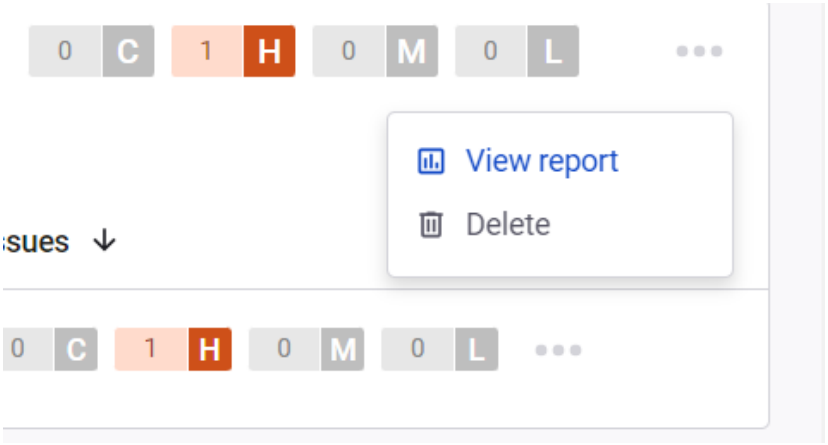


Рис. 3.38. Функціонал створення звітів

До того ж, для зручності пріоритизації наявних у вебзастосунку вразливостей, у Snyk існує спеціальний фільтр рівня серйозності знайдених небезпек. У межах цього фільтра всі вибрані значення розділяються логічним оператором OR (з англ. «АБО»). Наприклад, якщо вибрати значення "Критична" і "Висока" для фільтра "Серйозність проблеми", Snyk покаже проблеми, які мають серйозність або критичну, або високу.

Фільтри розділяються оператором AND. Наприклад, якщо вибрати значення «Критична» (рис. 3.39) для фільтра «Серйозність проблеми» і значення «Вирішено» для фільтра «Статус проблеми», Snyk покаже проблеми, які мають як критичну серйозність, так і вирішені [37].

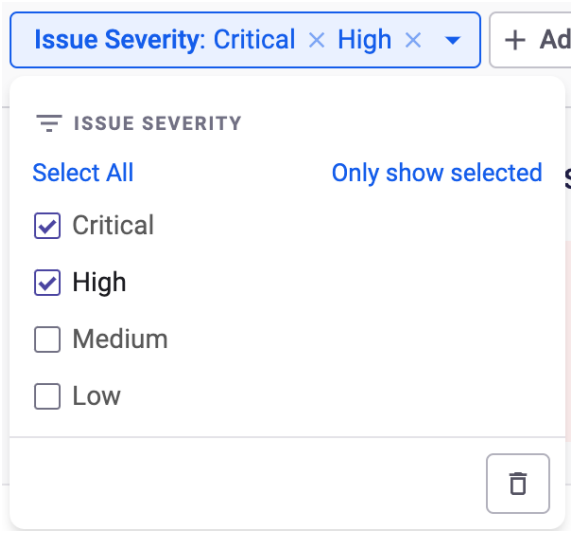


Рис. 3.39. Налаштування фільтрів звітності [37]

Звіти у Snyk можна переглядати у декількох різних форматах – у вигляді вебсторінки (рис. 3.40), у вигляді файла з розширенням .csv та/або .pdf.

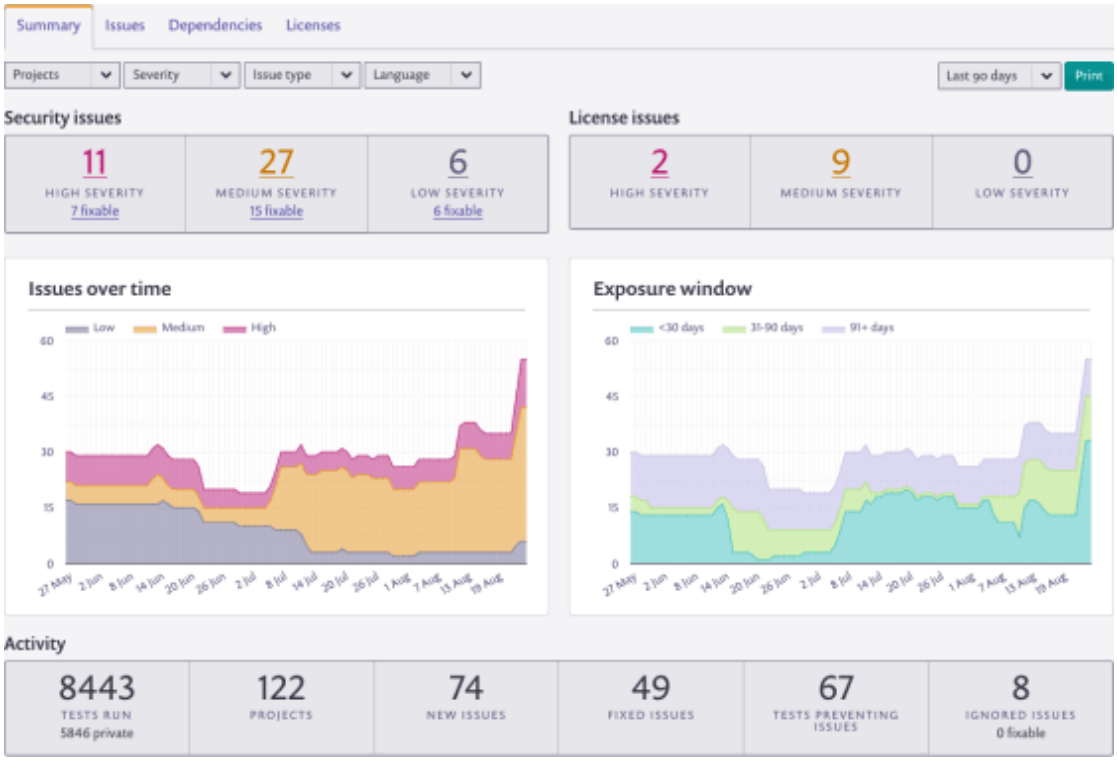
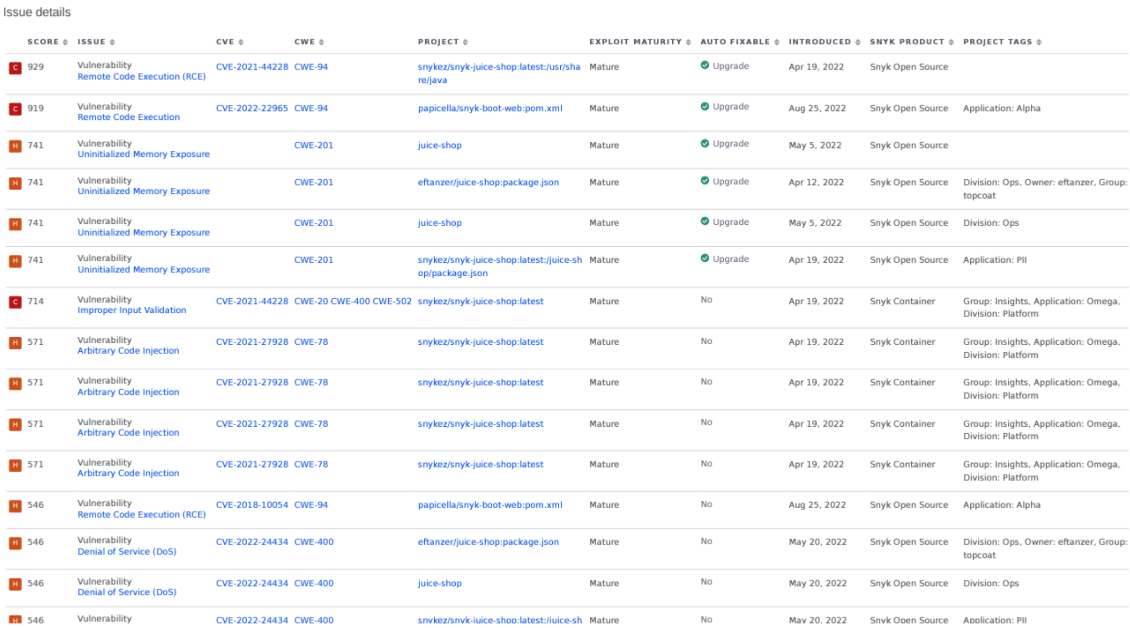


Рис. 3.40. Можлвивий вигляд вебзвіту [37]

Для створення звітів Snyk у форматах .csv і .pdf користувачі можуть скористатися кнопками "Експортувати в PDF" і "Завантажити CSV", які знаходяться в інтерфейсі звіту. Експорт у PDF включає вміст звіту, контекст і метадані, наприклад, хто і коли створив звіт, з обмеженням на показ лише перших 50 результатів. Цей формат підходить для обміну з особами за межами середовища програми Snyk. Приклад однієї зі сторінок такого звіту зображено на рис. 3.41.



Issue details

	SCORE	ISSUE	CVE	CWE	PROJECT	EXPLOIT MATURITY	AUTO FIXABLE	INTRODUCED	SNYK PRODUCT	PROJECT TAGS
C	929	Vulnerability Remote Code Execution (RCE)	CVE-2021-44228	CWE-94	snykez/snyk-juice-shop:latest:/usr/share/java	Mature	Upgrade	Apr 19, 2022	Snyk Open Source	
C	919	Vulnerability Remote Code Execution	CVE-2022-22965	CWE-94	papicella/snyk-boot-web:pom.xml	Mature	Upgrade	Aug 25, 2022	Snyk Open Source	Application: Alpha
H	741	Vulnerability Uninitialized Memory Exposure		CWE-201	juice-shop	Mature	Upgrade	May 5, 2022	Snyk Open Source	
H	741	Vulnerability Uninitialized Memory Exposure		CWE-201	eftanzer/juice-shop:package.json	Mature	Upgrade	Apr 12, 2022	Snyk Open Source	Division: Ops, Owner: eftanzer, Group: topcoat
H	741	Vulnerability Uninitialized Memory Exposure		CWE-201	juice-shop	Mature	Upgrade	May 5, 2022	Snyk Open Source	Division: Ops
H	741	Vulnerability Uninitialized Memory Exposure		CWE-201	snykez/snyk-juice-shop:latest:/juice-shop/package.json	Mature	Upgrade	Apr 19, 2022	Snyk Open Source	Application: PII
C	714	Vulnerability Improper Input Validation	CVE-2021-44228	CWE-20 CWE-400 CWE-502	snykez/snyk-juice-shop:latest	Mature	No	Apr 19, 2022	Snyk Container	Group: Insights, Application: Omega, Division: Platform
H	571	Vulnerability Arbitrary Code Injection	CVE-2021-27928	CWE-78	snykez/snyk-juice-shop:latest	Mature	No	Apr 19, 2022	Snyk Container	Group: Insights, Application: Omega, Division: Platform
H	571	Vulnerability Arbitrary Code Injection	CVE-2021-27928	CWE-78	snykez/snyk-juice-shop:latest	Mature	No	Apr 19, 2022	Snyk Container	Group: Insights, Application: Omega, Division: Platform
H	571	Vulnerability Arbitrary Code Injection	CVE-2021-27928	CWE-78	snykez/snyk-juice-shop:latest	Mature	No	Apr 19, 2022	Snyk Container	Group: Insights, Application: Omega, Division: Platform
H	571	Vulnerability Arbitrary Code Injection	CVE-2021-27928	CWE-78	snykez/snyk-juice-shop:latest	Mature	No	Apr 19, 2022	Snyk Container	Group: Insights, Application: Omega, Division: Platform
H	546	Vulnerability Remote Code Execution (RCE)	CVE-2018-10054	CWE-94	papicella/snyk-boot-web:pom.xml	Mature	No	Aug 25, 2022	Snyk Open Source	Application: Alpha
H	546	Vulnerability Denial of Service (DoS)	CVE-2022-24434	CWE-400	eftanzer/juice-shop:package.json	Mature	No	May 20, 2022	Snyk Open Source	Division: Ops, Owner: eftanzer, Group: topcoat
H	546	Vulnerability Denial of Service (DoS)	CVE-2022-24434	CWE-400	juice-shop	Mature	No	May 20, 2022	Snyk Open Source	Division: Ops
H	546	Vulnerability	CVE-2022-24434	CWE-400	snykez/snyk-juice-shop:latest:/juice-sh	Mature	No	May 20, 2022	Snyk Open Source	Application: PII

Рис. 3.41. Приклад однієї зі сторінок звіту у форматі .pdf [37]

Експорт у форматі CSV дозволяє проводити детальний аналіз в електронних таблицях, включаючи всі стовпці, що відображаються в інтерфейсі користувача, і розділення гіперпосилань, без обмеження на кількість рядків, але з обмеженням на розмір файлу в 5 ГБ.

Важливо зазначити, що у деяких звітах таблиці можуть містити опцію зміни стовпців. Якщо ця опція доступна, то можна використовувати її для вибору стовпців для відображення в інтерфейсі користувача. Функції експорту (PDF і CSV) враховують вибрані стовпці.

Коли стовпці змінюються, URL-адреса app.snyk.io оновлюється, щоб зберегти стан сторінки, що дозволяє робити закладки, копіювати та ділитися нею.

Додатково Snyk надсилає сповіщення на електронну адресу користувачів у разі знаходження вразливостей у коді проєктів. Це доволі зручне та клієнтоорієнтоване рішення для полегшення менеджменту процесу розробки вебзастосунків чи, в цілому, написання будь-якого коду. Приклад такого повідомлення зображено на рис. 3.42.

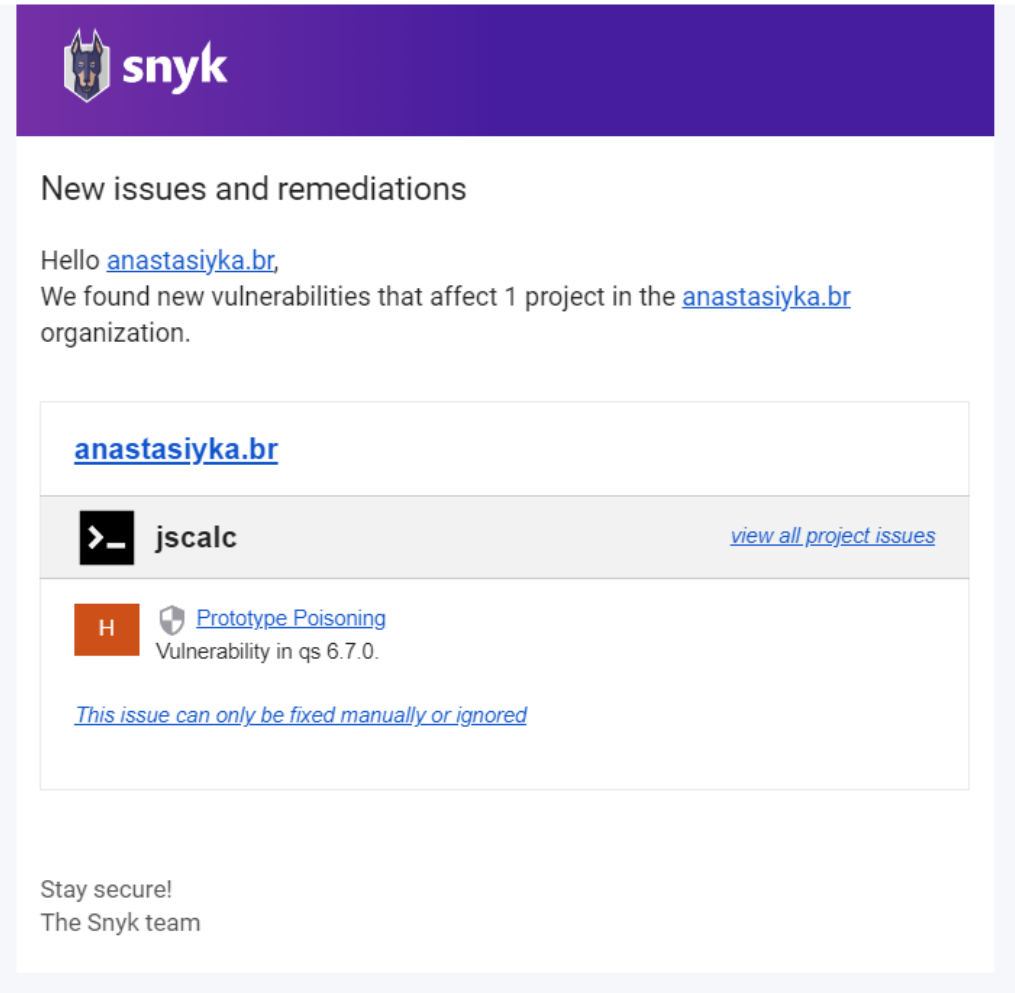


Рис. 3.42. Інформування про наявність вразливостей у вебзастосунку через електронну пошту

Також щотижня Snyk формує загальний звіт з проєктів, в якому показує загальну кількість знайдених за тиждень вразливостей, їхню серйозність, загальну кількість залежностей та містить посилання на перегляд вразливості у, безпосередньо, середовищі рішення. Приклад такого звіту можна побачити на рисунку 3.43.

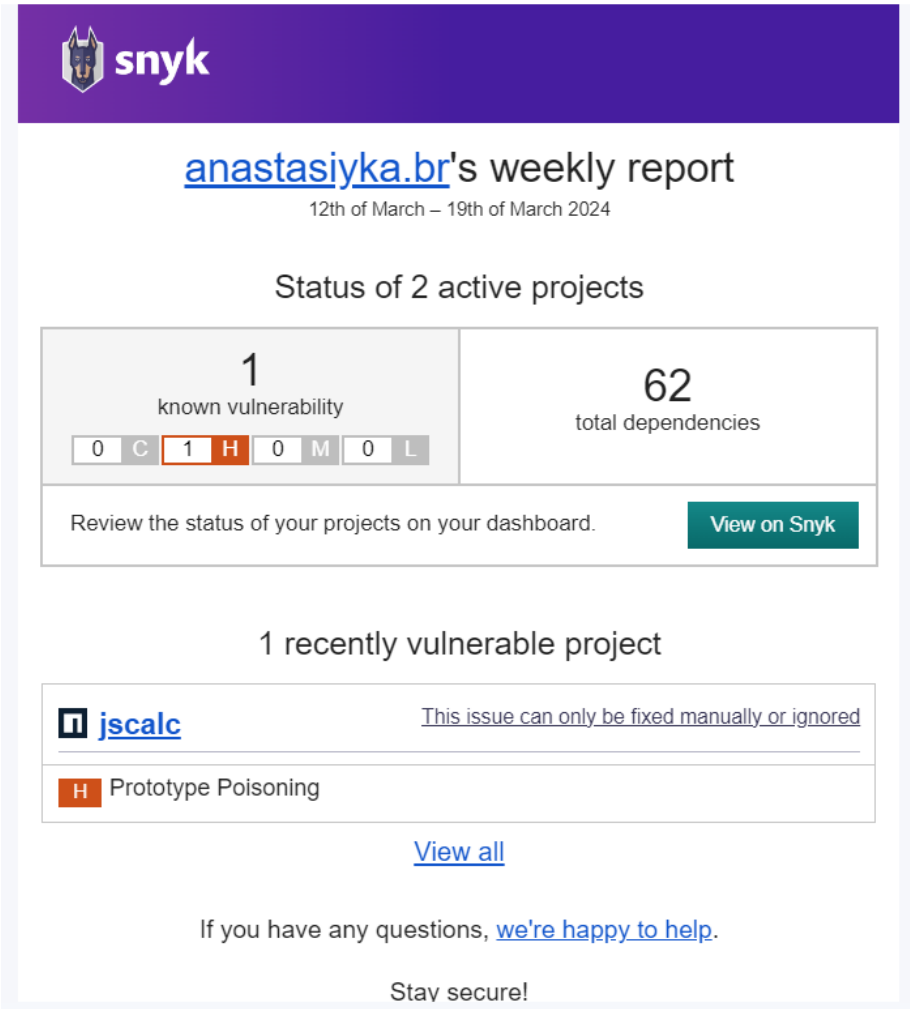


Рис. 3.43. Тижневий звіт у форматі повідомлення на електронній пошті

Отже, використання Snyk для генерації звітів у форматах .csv та .pdf значно покращує процес управління безпекою програмного забезпечення, надаючи комплексне уявлення про поточний стан безпеки проектів. Ці звіти детально описують вразливості, їх вплив та рекомендації щодо виправлення, дозволяючи командам ефективно планувати та виконувати дії зі зміцнення захисту. Функція експорту забезпечує легкість обміну цією критично важливою інформацією між членами команди, стимулюючи спільну роботу та своєчасне реагування на загрози. Таке інтегроване використання звітів Snyk підкріплює ідею про те, що ефективне управління безпекою програмного забезпечення вимагає постійного моніторингу та адаптації на кожному етапі життєвого циклу розробки.

3.5. Рекомендації щодо застосування методів та засобів виявлення вразливостей вебзастосунків

Розглянемо стратегії та інструменти, які максимально ефективно допоможуть ідентифікувати та усунути вразливості у вебзастосунках. Починаючи з огляду сучасних методів виявлення вразливостей, таких як статичний та динамічний аналіз коду, до використання фахівцями інструментів автоматизації на кшталт Snyk для глибокого аналізу залежностей та коду. Особлива увага повинна бути приділена важливості інтеграції цих процесів у ранніх етапах розробки та їх впливу на підвищення загального рівня безпеки програмного продукту.

Отже, для покращення безпеки вебзастосунків, варто розглянути наступні рекомендації:

Комбінація методології SAST і DAST для всебічного виявлення вразливостей. Така синергія є ключовою стратегією для досягнення всебічного виявлення вразливостей у вебзастосунках. SAST аналізує код на ранніх етапах розробки, виявляючи потенційні вразливості без виконання програми, тоді як DAST тестує запущені застосунки в реальному часі, імітуючи зовнішні атаки. Ця двоїста перевірка забезпечує глибокий аналіз безпеки, виявляючи вразливості, які можуть бути пропущені при використанні лише однієї методології. Комбінація цих підходів дозволяє розробникам ефективно ідентифікувати та усувати слабкі місця до розгортання продукту, значно підвищуючи загальний рівень безпеки застосунків.

Застосування рішення класу SAST на ранніх етапах розробки ПЗ як превентивний захід. Використання рішень класу SAST на ранніх етапах розробки програмного забезпечення дозволяє розробникам виявляти вразливості та проблеми безпеки ще до того, як код потрапляє у безпосереднє робоче середовище. Цей підхід не тільки зменшує ризики безпеки для кінцевого продукту, але й значно знижує витрати на виправлення помилок, оскільки усунення вразливостей на ранніх стадіях є менш витратним за часом та ресурсами. Проактивне застосування

SAST сприяє підвищенню загального рівня безпеки розроблюваних систем, створюючи надійну основу для подальшої роботи над проєктом.

Інтеграція Snyk з різними IDE для спрощення процесу безпечної розробки, дозволяючи командам розробників виявляти та усувати вразливості безпосередньо під час написання коду. Ця функціональність забезпечує безперервний моніторинг коду на наявність потенційних вразливостей і зловмисних залежностей, автоматизуючи процеси безпеки та зменшуючи ризик введення вразливостей у фінальний продукт. Використання Snyk у поєднанні з улюбленими IDE розробників, такими як Visual Studio Code, IntelliJ IDEA, або Eclipse, робить процес інтеграції безпеки природною частиною розробки, підвищуючи якість коду та ефективність робочих процесів.

Автоматизація сканування при кожному значному оновленні коду або при кожному коміті. Це є важливим кроком у підтримці безпеки програмного забезпечення. Таке рішення дозволяє командам швидко ідентифікувати та реагувати на потенційні ризики безпеки в реальному часі, знижуючи ймовірність проникнення вразливостей в середовище функціонуванняіі. Особливо корисним цей підхід є в середовищах, де практикується безперервна інтеграція та безперервне розгортання (CI/CD), оскільки він допомагає підтримувати високий рівень безпеки на всіх етапах розробки.

Ретельне слідування інструкціям з виправлення вразливостей. Після ідентифікації вразливостей за допомогою інструментів, таких як Snyk, необхідно детально вивчити кожну рекомендацію, яка надається разом з результатами сканування. Це включає аналіз впливу вразливостей, пріоритизацію залежно від їх критичності та виконання рекомендованих кроків для їх усунення. На практиці це може означати оновлення до безпечних версій бібліотек, зміну конфігурацій або модифікацію коду для усунення слабких місць. Ключовим аспектом є здатність швидко реагувати на вразливості, оскільки затримка може збільшити ризик витоку даних чи інших безпекових інцидентів.

Неперервне навчання команди розробників безпечним практикам кодування для створення міцної культури безпеки. Неперервне навчання команд розробників

практикам безпечного кодування є важливим кроком для створення міцної культури безпеки в організації. Цей процес передбачає регулярні тренінги, воркшопи та семінари, які охоплюють актуальні теми кібербезпеки, найновіші вразливості та методи їх усунення. Підвищуючи рівень обізнаності розробників, організації можуть значно знизити ризики безпеки на етапі кодування, сприяючи розробці більш безпечних продуктів.

Отже, у цьому розділі було розглянуто різноманітні підходи та інструменти для виявлення та усунення вразливостей у веб-застосунках. Висвітлено важливість комбінування статичного і динамічного аналізу, наголошено на користі використання інструментів, таких як Snyk Code, для спрощення цих процесів. Підкреслено, що автоматизація сканувань та неперервне навчання команд розробників мають стати інтегральною частиною процесу розробки, адже це дозволяє підвищити загальний рівень безпеки продукту та мінімізувати ризики.

ВИСНОВКИ

Проведено аналіз сучасних методів виявлення вразливостей у вебзастосунках, що є критично важливим для забезпечення кібербезпеки.

Розглянуто шляхи інтеграції статичного аналізу безпеки застосунків (SAST) з використанням рішення Snyk, яке демонструє високу ефективність у ранньому виявленні потенційних загроз.

Встановлено значні переваги комбінованого застосування методів SAST і DAST для всебічного аналізу безпеки вебзастосунків, а також важливість автоматизації процесів сканування коду для оперативного реагування на зміни.

Рекомендації, розроблені на основі дослідження, спрямовані на підвищення ефективності процесів розробки та забезпечення безпеки вебзастосунків, включають необхідність неперервного навчання розробників, інтеграцію безпечних практик кодування, та використання інноваційних рішень методології SAST на кшталт Snyk. Реалізація таких рекомендацій дозволить організаціям покращити свої стратегії кіберзахисту, мінімізувати ризики, та ефективно реагувати на постійно змінювані кіберзагрози.

Здобуті результати дослідження вказують на ключові напрямки для вдосконалення кібербезпеки, підкреслюючи значення створення захищених вебзастосунків. Результати пропонують наукове та практичне використання, сприяючи розробці надійних програмних рішень. Важливим фактором є імплементація передових методів виявлення вразливостей у коді програмного забезпечення, що сприятиме підвищенню загального рівня безпеки вебзастосунків.

ПЕРЕЛІК ПОСИЛАНЬ

1. Why Is Web Application Development Valuable for Businesses?. URL: https://www.linkedin.com/pulse/why-web-application-development-valuable-businesses-wp6we/ (date of access: 21.02.2024).

2. Facts and Factors. Study on Global Progressive Web Application Market Size to Hit US$ 7,600 Mn, at a CAGR of 34% by 2026. Facts and Factors. URL: https://www.fnfresearch.com/news/global-progressive-web-application-market-revenue-projected-around (date of access: 21.02.2024).

3. 7 Benefits of Developing a Web-based application for Your Business. SmartTek Solutions. URL: https://smarttek.solutions/blog/web-based-app-for-businesses/ (date of access: 23.02.2024).

4. Pranto D. Ztos- HR Management System B2B Webapp. Dribbble. URL: https://dribbble.com/shots/14268458-Ztos-HR-Management-System-B2B-Webapp (date of access: 23.02.2024).

5. What is Web Application (Web Apps) and its Benefits. TechTarget. URL: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app (date of access: 23.02.2024).

6. Oza H. The Impact of Web App on Your Business | Hyperlink InfoSystem. Top Mobile App Development Company USA, India & UAE - Hyperlink InfoSystem. URL: https://www.hyperlinkinfosystem.com/blog/impact-of-web-app-development-on-your-business (date of access: 26.02.2024).

7. 6 основних етапів SDLC. QualityAssuranceGroup. URL: https://qagroup.com.ua/publications/6-osnovnykh-etapiv-sdlc/ (дата звернення: 27.02.2024).

8. Aytemiz T. How does the SDLC play a role in the success of product development and launch?. Medium. URL: https://medium.com/agileinsider/how-does-the-sdlc-play-a-role-in-the-success-of-product-development-and-launch-a17baaac1054 (date of access: 29.02.2024).

9. Zaigo Infotech Software Solutions Pvt Ltd. Top 7 benefits of Web Application Development for Business?. LinkedIn. URL: https://www.linkedin.com/pulse/top-7-benefits-web-application-development/ (date of access: 01.03.2024).

10. Trofymenko O., Dyka A., Loboda Y. Analysis of vulnerabilities and security problems of web applications. System technologies. 2023. Vol. 3, no. 146. P. 25–37. URL: https://doi.org/10.34185/1562-9945-3-146-2023-03 (date of access: 01.03.2024).

11. Що таке OWASP? ТОП 10 вразливостей | Онлайн-курси від компанії QATestLab. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/what-is-owasp-top-10-vulnerabilities/ (дата звернення: 06.03.2024).

12. OWASP Top 10:2021. OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. URL: https://owasp.org/Top10/ (date of access: 07.03.2024).

13. Shneider T. Application security testing: types, tech, and 5 critical best practices. Pynt.io. URL: https://www.pynt.io/guides/application-security-testing-guide/application-security-testing#heading-3 (date of access: 08.03.2024).

14. Yosef G. DAST vs. SAST. Pynt.io. URL: https://www.pynt.io/guides/application-security-testing-guide/dast-vs-sast#heading-1 (date of access: 09.03.2024).

15. What is Software Application Security Testing (SAST)?. Sonatype. URL: https://www.sonatype.com/launchpad/what-is-sast (date of access: 09.03.2024).

16. Source Code Analysis Tools. OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (date of access: 10.03.2024).

17. Nair M. Snyk named a Leader in 2023 Gartner® Magic Quadrant™ for Application Security Testing | Snyk. Snyk. URL: https://snyk.io/blog/snyk-named-leader-2023-magic-quadrant/ (date of access: 10.03.2024).

18. Static Code Analysis | Veracode. Veracode. URL: https://www.veracode.com/security/static-code-analysis (date of access: 11.03.2024).

19. SAST Scan: Source Code Vulnerability Scanner - Checkmarx.com.

Checkmarx. URL: https://checkmarx.com/cxsast-source-code-scanning/ (date of access: 11.03.2024).

20. SAST - Static Code Analysis Tools. Synopsys. URL: https://www.synopsys.com/software-integrity/static-analysis-tools-sast.html (date of access: 11.03.2024).

21. Code Security Analysis and Fixes - Developer First SAST | Snyk. Snyk. URL: https://snyk.io/product/snyk-code/ (date of access: 11.03.2024).

22. Getting started. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/getting-started (date of access: 12.03.2024).

23. Integrate with Snyk. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/integrate-with-snyk (date of access: 12.03.2024).

24. Use Snyk in your IDE | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/integrate-with-snyk/ide-tools (date of access: 12.03.2024).

25. Snyk CI/CD integrations | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/integrate-with-snyk/snyk-ci-cd-integrations (date of access: 13.03.2024).

26. Supported languages and frameworks | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/getting-started/supported-languages-frameworks-and-feature-availability-overview (date of access: 13.03.2024).

27. Snyk Learn | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/getting-started/snyk-learn (date of access: 13.03.2024).

28. Luz L. Benefits of Using Snyk for Static Application Security Testing (SAST). Medium. URL: https://luanluz.medium.com/benefits-of-using-snyk-for-static-application-security-testing-sast-2ca0acfbbb01 (date of access: 14.03.2024).

29. Implement Snyk | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/implement-snyk (date of access: 14.03.2024).

30. Scan with Snyk | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/scan-with-snyk (date of access: 14.03.2024).

31. Hack The Box. Hack The Box. URL: https://app.hackthebox.com/ (date of

access: 15.03.2024).

32. jscalc. Hack The Box. URL: https://app.hackthebox.com/challenges/jscalc (date of access: 15.03.2024).

33. What is prototype pollution? | Tutorial & examples | Snyk Learn. Snyk Learn. URL: https://learn.snyk.io/lesson/prototype-pollution/ (date of access: 15.03.2024).

34. Top 20 Best VScode Extensions for 2024. Jit. URL: https://www.jit.io/blog/vscode-extensions-for-2023 (date of access: 18.03.2024).

35. Snyk Security - Visual Studio Marketplace. Extensions for Visual Studio family of products | Visual Studio Marketplace. URL: https://marketplace.visualstudio.com/items?itemName=snyk-security.snyk-vulnerability-scanner (date of access: 19.03.2024).

36. Visual Studio extension configuration | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/integrate-with-snyk/use-snyk-in-your-ide/visual-studio-extension/visual-studio-extension-configuration (date of access: 19.03.2024).

37. Introducing Snyk Reports | Snyk User Docs. Snyk Documentation | Snyk User Docs. URL: https://docs.snyk.io/manage-risk/reporting/getting-started-with-snyk-reports (date of access: 20.03.2024).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)